Dossier technique

Periscol

Réalisé par:

Baptiste Cahuziere - Cédric Cognard - Clément Peytureau - Dorian Caup - Romain Gaillard

Version numérique :
http://urlz.fr/puD

Lycée A. KASTLER

SOMMAIRE

I. Présentation du projet.....6
1) Objectif.....6
2) Contexte.....6
3) Enjeux pour le demandeur.....7
4) Résultats attendus.....7
5) Fonctionnalité et contraintes.....7
II. Spécifications générales.....10
1) Organisation du projet.....10
2) Diagramme de cas d'utilisation.....11
3) Diagramme de paquetage.....12
4) Diagramme de séquence général.....13
III. Description de l'architecture.....14
1) Diagramme de nœuds.....14
IV. Répartition du travail.....14
V. Description de la partie de Dorian Caup – Serveur Jboss.....16
1) Description Web Service JBoss :.....16
2) Mise en œuvre :.....17
a. Le protocole SOAP :.....17
b) Environnement de travail :.....18
2) Conception détaillée:.....19
a) Diagramme de classe détaillé :.....19
b) Diagramme de séquence détaillé : CreerCompte() :.....20
c) Diagramme de séquence détaillé : MdpOublie(String) :.....20
d) Diagramme de séquence détaillé AjouterEnfant(String, String, bool, String, ArrayList<String>, String, int) :.....20
e) Diagramme de séquence détaillé SeConnecter(String,String) :.....21
3) Implémentation :.....21
a) Notions clés sur la technologie utilisée 22
a.2. javax.xml.bind.annotation.XmlType :.....23
a.3. JDBC : Java Data Base Connector :.....24
a.4. Petit service supplémentaire :.....25
b) Étude du code Mdp_oublie(String) :.....26
c) Étude du code Get_All_Service(String) :.....27
4) Test de validation :.....28
5) Bilan personnel :.....30
VI. Description de la partie Clément Peytureau - Gestionnaire.....30
1) Description de la partie personnelle.....30
2) Mise en œuvre.....30

a. Fonctionnement de l'application....30
b. Comment mettre en œuvre ?....30
b.1. Installation de JavaFX22....30
b.2. JavaFX2 SceneBuilder....31
b.3. Intégration SceneBuilder à Eclipse....31
b.4. Intégration de la librairie Axis2 1.6.2....32
b.5. Fonctionnement du protocole SOAP....33
3) Conception détaillée....34
a. Diagramme de cas d'utilisation....34
b. Diagramme de classe détaillé....34
c. Diagramme séquence AfficherStat....36
d. Diagramme de séquence SupprimerCompte....36
e. Diagramme de séquence AjouterService....37
f. Diagramme de séquence ModifService....37
g. Diagramme de séquence SupprimerService....38
4) Implémentation....38
a. Principe de fonctionnement....38
b. Authentification....40
5) Test de validation....45
6) Bilan personnel....45
VII. Partie personnelle – Baptiste CAHUZIERE – Base de données....46
1) Description de la partie personnelle....46
2) Mise en œuvre....46
a. Installation de la base de données....46
b. Interactions avec le serveur....49
c. Fonctionnement du protocole JDBC....49
3) Conception détaillée....50
b. Diagramme de cas d'utilisation....50
c. Diagramme de classes détaillé....51
d.Diagramme de séquence....52
4) Implémentation....53
a- Connexion à la base de données....53
b- Exécution d'une requête....53
5) Test de validation....54
a- Connexion à la base de données....54
b- Exécution d'une requête....54
6)Bilan personnel....57
VIII. Partie personnelle de Romain Gaillard – Usager Android 58
1) Description de la partie personnelle....58

2) Mise en Œuvre 58
a. Fonctionnement de l'application 58
b. Environnement de développement 58
c. Protocole de communication SOAP 60
c.1 Schéma de fonctionnement : *60*
c.2 Détail du fonctionnement : *60*
3) Conception détaillée (diagrammes) 61
a. Diagramme de cas d'utilisation 61
b. Diagramme de classe détaillé 61
c. Diagramme de séquence (Inscription) 63
d. Diagramme de séquence (Consulter les Services) 64
e. Diagramme de séquence (Mot de passe oublié) 65
f. Diagramme de séquence (Connexion) 66
h. Diagramme de séquence (Ajouter Enfant) 67
i. Diagramme de séquence (Modifier Enfant) 68
j. Diagramme de séquence (Supprimer Enfant) 69
k.Diagramme de séquence (Afficher les Enfants) 70
l. Diagramme de séquence (Modifier Compte) 71
4) Implémentation 72
a. IHM Connexion 72
a.1 Classe Client_cli : *75*
a.2 Requête SOAP : *75*
a.3 La méthode Synchro *76*
a.4 Consulter les services *76*
a.5 Récupération de mot de passe *78*
b. IHM Inscription 79
c. IHM Accueil (Connecté) 81
d. IHM Enfant (Ajouter) 82
e. IHM Modifier Enfant 84
f. IHM Paramètres (du Compte) 84
h. Bonus 84
h.1 Les bases de la programmation Android *84*
h.2 Lancer une nouvelle activité *85*
h.3 Passage de paramètres entre activités *85*
h.4 Le scrolling *85*
6) Test de validation 86
a. Tests techniques 86
a. 1 Connexion *87*
a. 2 Inscription *88*

a. 3 Consulter les services.....89
b. Test fonctionnels.....90
7) Bilan personnel 92
IX. Partie personnelle de Cédric Cognard – Usager iOs:.....93
1) Description de la partie personnelle.....93
2) Mise en œuvre :.....93
a. Découverte Objective-C.....93
d. Web Service.....95
e. Aide à la mise en place du web service (SoapUI).....96
3) Conception détaillée :.....97
a. Diagramme de classes détaillé.....97
4) Implémentation :.....98
a.1. Champs textes :.....98
a.2. Connexion :.....99
a.3. Affichage des services :.....100
a.4. Bouton Inscription :.....100
b. Écran Inscription :.....101
5) Test de validation :.....102
1) Authentification :.....102
6) Bilan personnel :.....103
X. Conclusion générale:.....104

Remerciements

Nous voulons remercier l'ensemble des enseignants qui nous ont assistés et dirigés durant ce projet de fin de formation, c'est-à-dire M. Lajus, M. Speziali, M. Perruche et particulièrement M. Perez, notre chef de projet, qui nous a permis de découvrir un projet captivant et qui a fait en sorte que celui-ci aboutisse.

Nous remercions aussi la société e-Citiz, ainsi que M. Olivier Nicolas, directeur de la société, pour nous avoir fourni ce projet tout particulièrement intéressant.

I. Présentation du projet

1) Objectif

Le projet **e-PeriScol** consiste en la création d'un accès Web à des services périscolaires, sportifs et culturels, proposés par une mairie pour les enfants des usagers de la commune. Il doit aussi permettre à la mairie d'avoir des statistiques de fréquentation des services périscolaires, consultables via un ordinateur présent dans les locaux .

Par ailleurs, le but est de proposer une application mobile, sur les systèmes Android et iOS qui représente 80 à 90 % du marché du smartphone, ce choix est judicieux, car il touche un maximum de personne.

2) Contexte

Ce projet est proposé par la société **e-Citiz**, (http://www.e-citiz.com), qui développe un moteur d'exécution de télé-procédures pour l'e-administration des institutions gouvernementales et des collectivités locales.

Désignation du projet	Web Services Périscolaires
Référence	e-PeriScol
Maîtrise d'ouvrage	e-Citiz
Délégué à la maîtrise de l'ouvrage	Olivier NICOLAS (Directeur de e-Citiz)
Maîtrise d'œuvre du prototypage	Section de BTS IRIST – Lycée A. Kastler
Responsable de la maîtrise d'œuvre	Jean-Paul-Perez – Professeur d'informatique et réseau
Équipe de réalisation	5 étudiants
Budget	Néant (Open Source)
Objet	Création et utilisation de Web Services

3) Enjeux pour le demandeur

Les télé-procédures permettent aux mairies d'offrir des services en ligne à leurs usagers, comme l'inscription à la cantine scolaire, à un centre aéré, la demande d'une aide sociale, le paiement en ligne de factures etc.

Les services de mairie sont intéressés par la proposition d'activités périscolaires pour les enfants, notamment sportives et culturelles. Ils souhaitent aussi pouvoir établir des statistiques diverses concernant leur utilisation, afin de les améliorer ou d'en élargir le catalogue.

Les services de mairie veulent que les inscriptions aux activités périscolaires puissent s'effectuer par les usagers à partir d'un PC ou d'un dispositif de saisie nomade *(smartphone, tablette tactile)*.

4) Résultats attendus

Le projet **e-PeriScol** devra permettre à:

- un **usager** d'inscrire ou de désinscrire son enfant à une activité périscolaire sportive ou culturelle, à l'aide de tout poste client *(ordinateur, tablette tactile, smartphone).*

- un **gestionnaire**, employé de mairie, d'établir des statistiques sur l'utilisation des activités périscolaires, depuis un poste situé sur le réseau de la marie ou à distance, par Internet.

5) Fonctionnalité et contraintes

Réalisation des services sous la forme de Web Services (protocole SOAP - Simple Object Access Protocol) avec serveur Java.

Pour les postes clients des usagers :
- Tablettes tactiles ou smartphones sous Android (3.2 Honeycomb et plus)
- iPhone (iOS 5 et plus)

Pour le poste gestionnaire : Ordinateur de bureau sous Windows 7

Détail des cas d'utilisation et contraintes associées :

Cas d'utilisation	Description	Contraintes
Se Connecter	S'identifier à l'aide d'un identifiant et d'un mot de passe	Les usagers sont censés être déjà inscrits dans la BD Usagers & Services, avec une adresse familiale, le nombre d'enfants vivant à cette adresse, avec pour chaque enfant le sexe et la date de naissance.
Consulter les services périscolaires	Obtenir la liste des services périscolaires, avec pour chaque service le jour et les heures de fonctionnement	Les services périscolaires sont censés être déjà notés dans la BD Usagers & Services. On considérera : - 3 services sportifs : judo, basket ball, football - 3 services culturels : percussions, dessin, poterie

Inscrire enfant à service périscolaire	Associer un enfant à un service périscolaire	L'enfant sera identifié par son nom, son prénom, son âge, son sexe, sa date de naissance, son adresse, l'identifiant familiale et sera inscrit dans la BD Usagers et Services. L'âge minimal d'inscription est 5 ans et l'âge maximal 18 ans. La date d'inscription sera notée.
Afficher les services périscolaires suivis par un enfant	Lister les services périscolaires	La requête se fait par le nom et le prénom de l'enfant. Les seuls services affichés sont ceux d'un enfant de la personne qui fait la requête.
Désinscrire enfant à service périscolaire	Dissocier un enfant et un service périscolaire.	L'enregistrement d'un enfant pour un service périscolaire sera supprimé. Les dates de désinscription sont notées de façon anonyme
Calculer statistiques utilisation par service périscolaire	Il s'agit de fournir pour chaque service : - Le nombre total de garçons et de filles inscrits - Le pourcentage d'enfants inscrits par rapport au nombre d'enfants inscriptibles - Le pourcentage de désinscriptions par mois glissant	Utilisation de JDBC pour obtenir les données de calcul.
Afficher statistiques utilisation par service périscolaire	Afficher les résultats précédents pour chaque service.	

<u>Liste des modifications apportées au cahier des charges :</u>

→ Cf. §IV pour les noms des étudiants.

- le gestionnaire hérite des fonctionnalités de l'usager *(Étudiant 2)*
- mise en place d'un trombinoscope *(Tous les étudiants)*
- gestion des mots de passes oubliés *(Usagers Étudiant 4&5, Gestionnaire Étudiant 2)*
- Gérer les activités périscolaires proposées. (Ajout, Suppression, Modification) avec modification possible de l'age minimum et maximum ainsi que l'ajout d'un nombre max de participant selon les services. (Gestionnaire – Étudiant 2 mais prise en compte des nouvelles caractéristiques d'un service par tous les étudiants).
- recherche d'un compte ou d'un enfant par le gestionnaire (*Étudiant 2)*
- supprimer un compte par le gestionnaire *(Étudiant 2)*

- cryptage des mots de passe *(Étudiants 1 & 3)*
- récupération de mot de passe *(Étudiants 1,4,5)*

Plate-forme de développement & Réalisation

Les plate-formes logicielles de développement des IHM seront :

- Sdk **Android** 3.2 (Honeycomb) et plus pour les tablettes tactiles
- Simulateur de développement pour iPhone sous **iOS** 5.0 et plus
- **Éclipse**, Java, **Java FX2** et SceneBuilder pour le client riche du poste de travail du gestionnaire sous Windows 7

La plate-forme de développement de la partie serveur d'application sera Éclipse, Java et **JDBC** pour l'accès à la base de données.

Le serveur d'application sera **JBoss**, avec Apache Axis2/Java pour les Web Services.
Le SGBD sera **MySQL**.

La modélisation du service doit être réalisée avec la méthode UML pour la spécification, la conception et le modèle d'implémentation.
Elle assurera un passage traçable à l'implémentation (sans génération automatique de code).
Pour la modélisation UML, **MagicDraw** sera utilisé.
Les sources seront mis en configuration avec **SVN**.

Contrainte économique

Les développements sont à base de technologies open source.

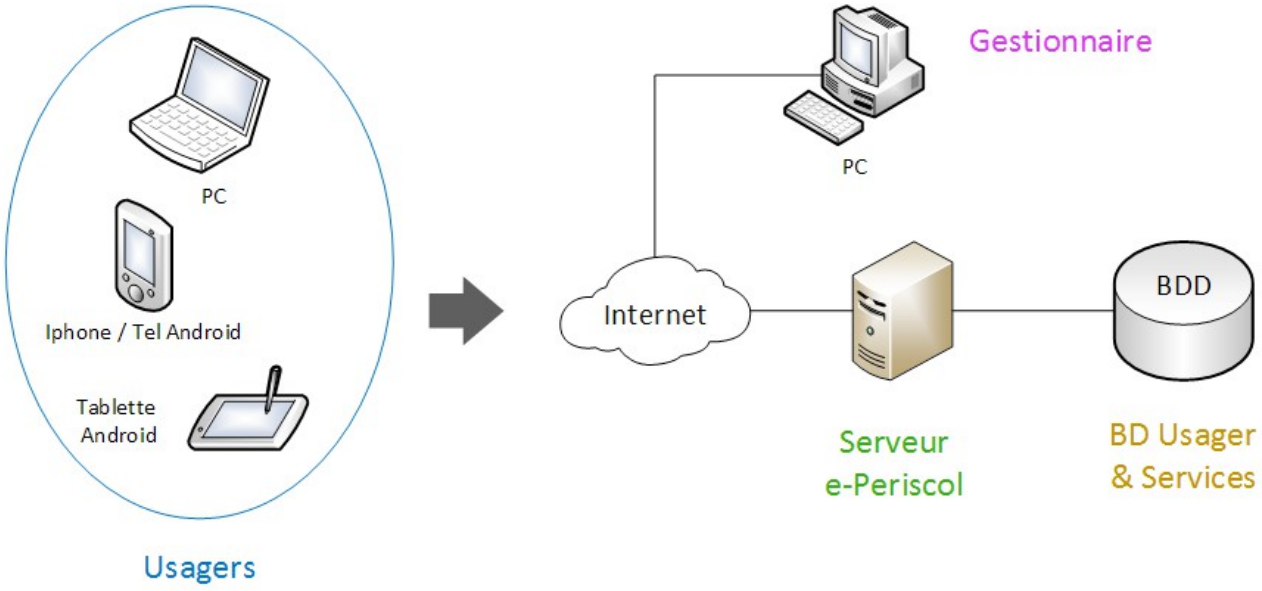
Exigences qualité sur le produit à réaliser

Le logiciel du système doit être :

- **maniable**, c'est-à-dire facile d'emploi avec IHM intégrée ;
- **robuste**, en conservant un fonctionnement conforme aux besoins lors d'une reprise après arrêt normal ou d'urgence;
- **maintenable**, en offrant une grande facilité de localisation et de correction des erreurs résiduelles, d'ajout ou de retrait de fonctionnalités;
- **portable** en environnement Unix Posix.

II. Spécifications générales

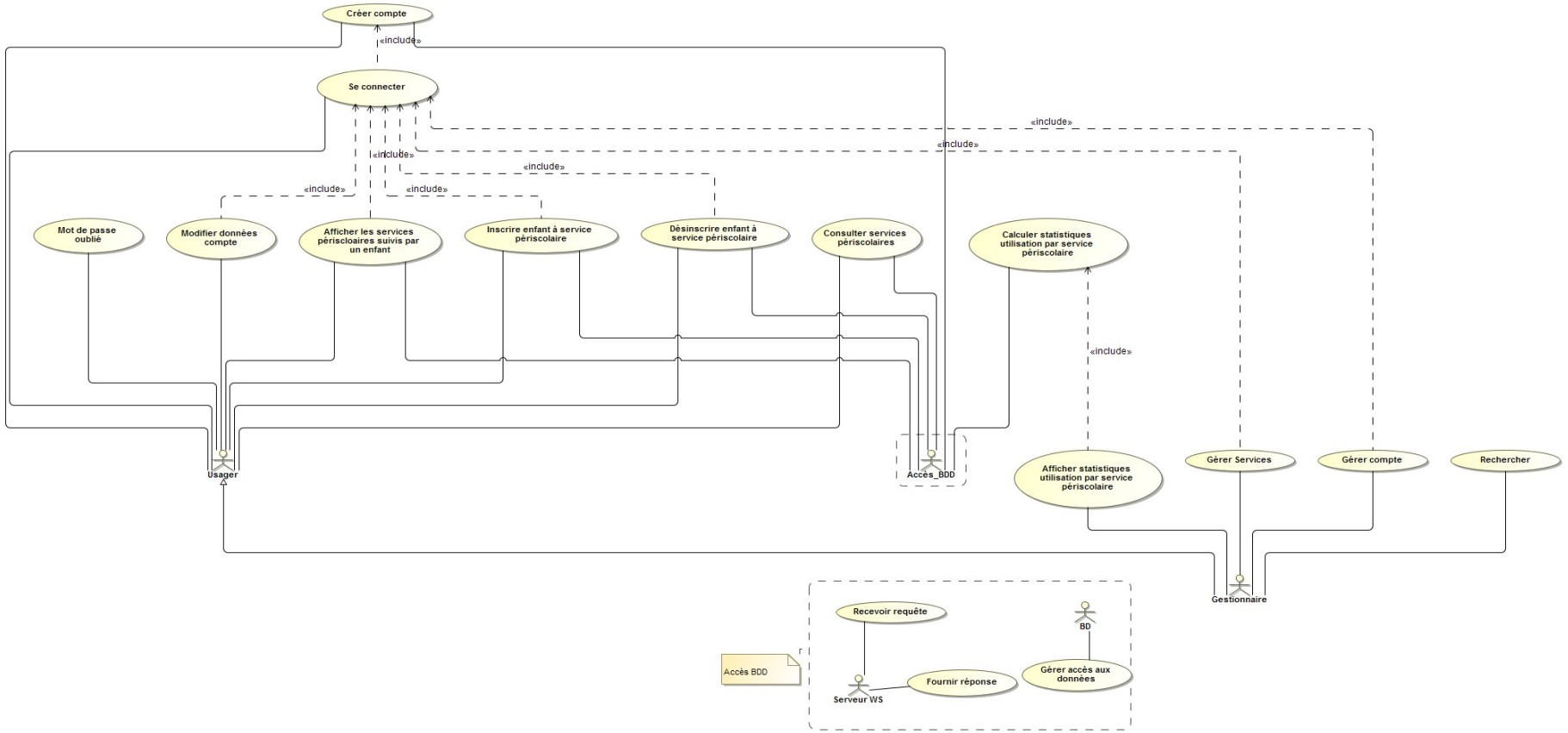
1) Organisation du projet



Les utilisateurs doivent pouvoir utiliser l'application depuis leurs tablettes ou smartphone *(Android ou iOS)* depuis le réseau Wi-fi pour se connecter à leur compte e-PeriScol via un Web Service situé sur le serveur Jboss.

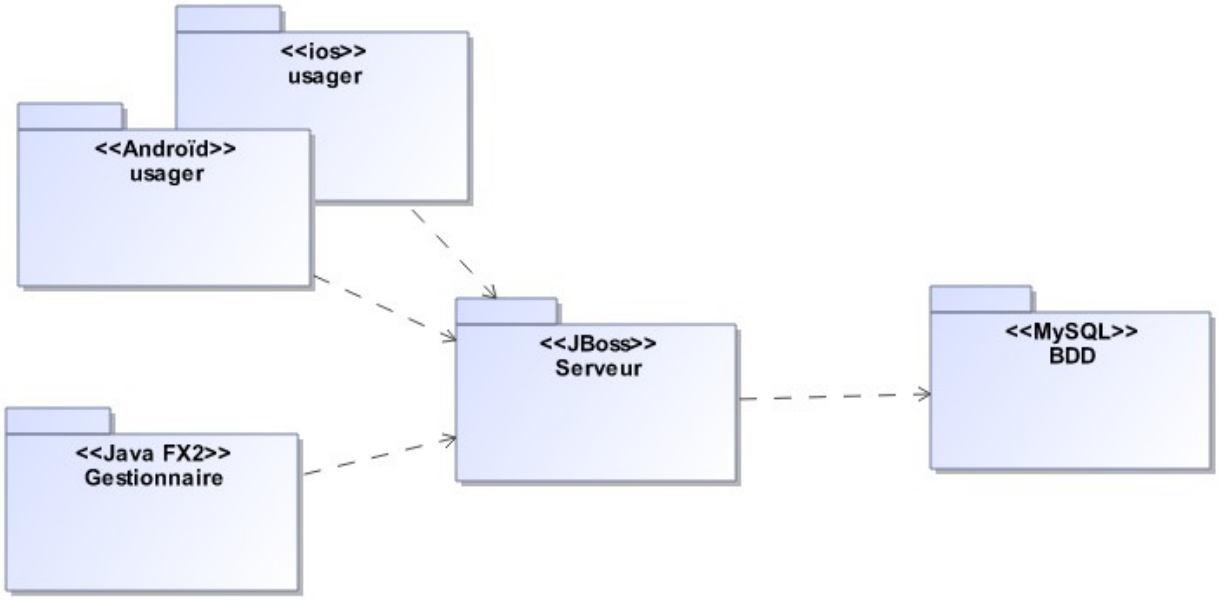
Ce projet est organisé avec une architecture trois tiers, cette organisation a comme avantage de séparer les clients de la base de données par un serveur. Les clients n'accèdent pas directement à la base de données, par conséquence ils n'effectuent pas de requêtes lourdes qui le font ralentir. De plus, les calculs sont fait par le serveur ce qui permet d'avoir un client plus léger.
Pour finir, cette architecture permet également au serveur de vérifier les données envoyées par le client et limite ainsi les erreurs possibles.

2) Diagramme de cas d'utilisation

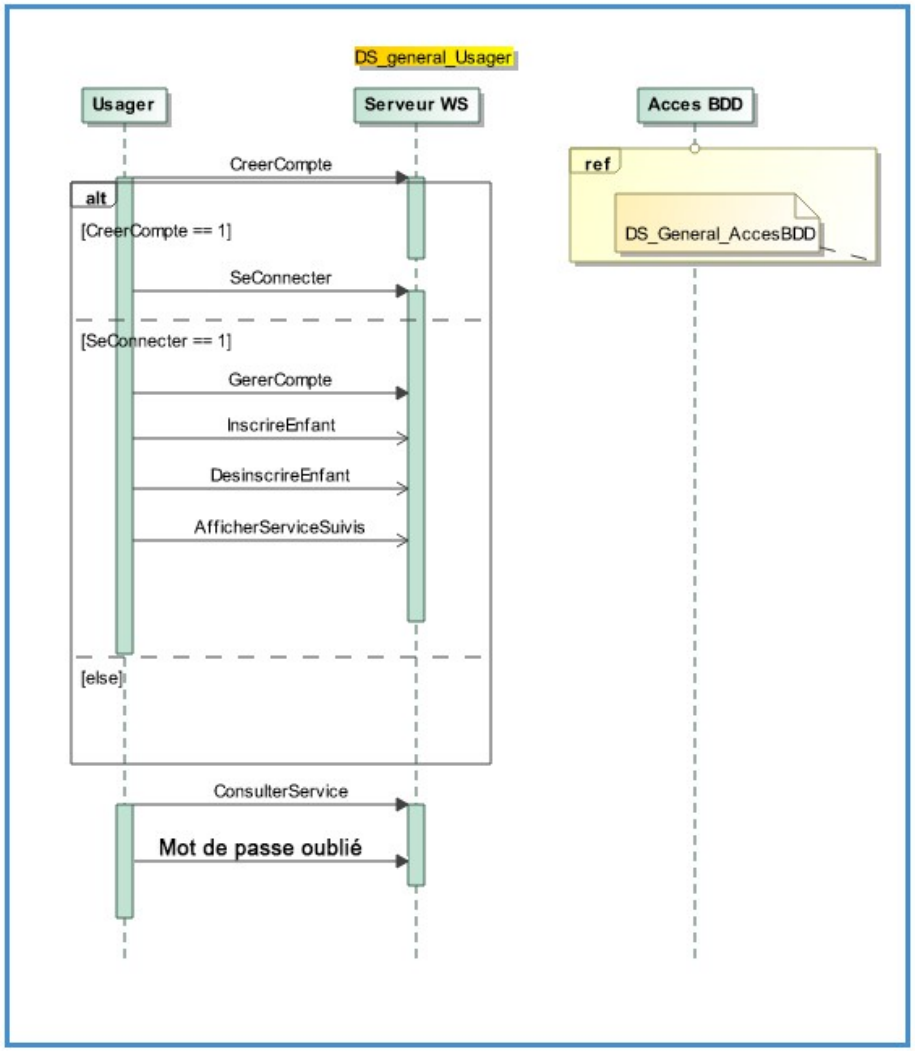


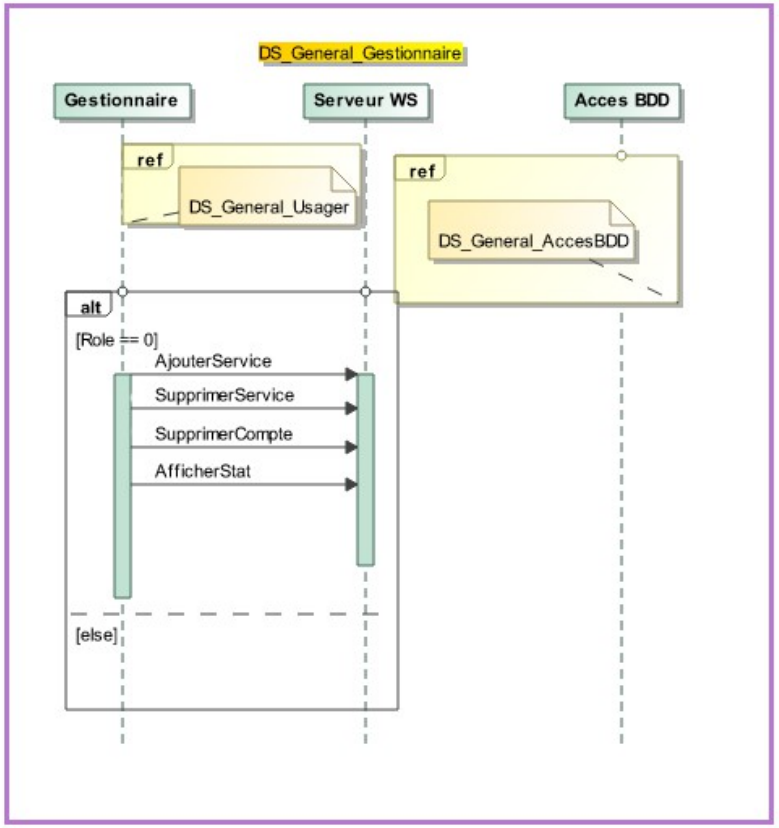
Haute qualité: http://image.noelshack.com/fichiers/2014/22/1401298399-dcu-general.png

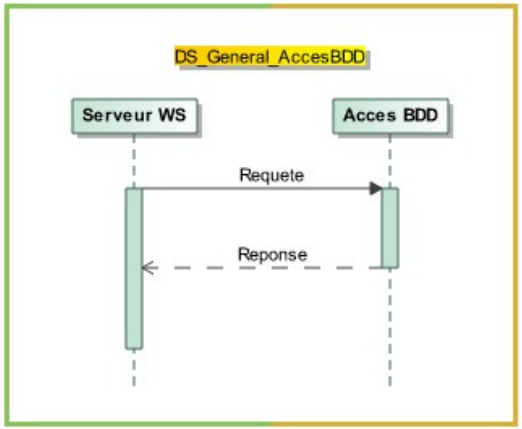
3) Diagramme de paquetage



4) Diagramme de séquence général

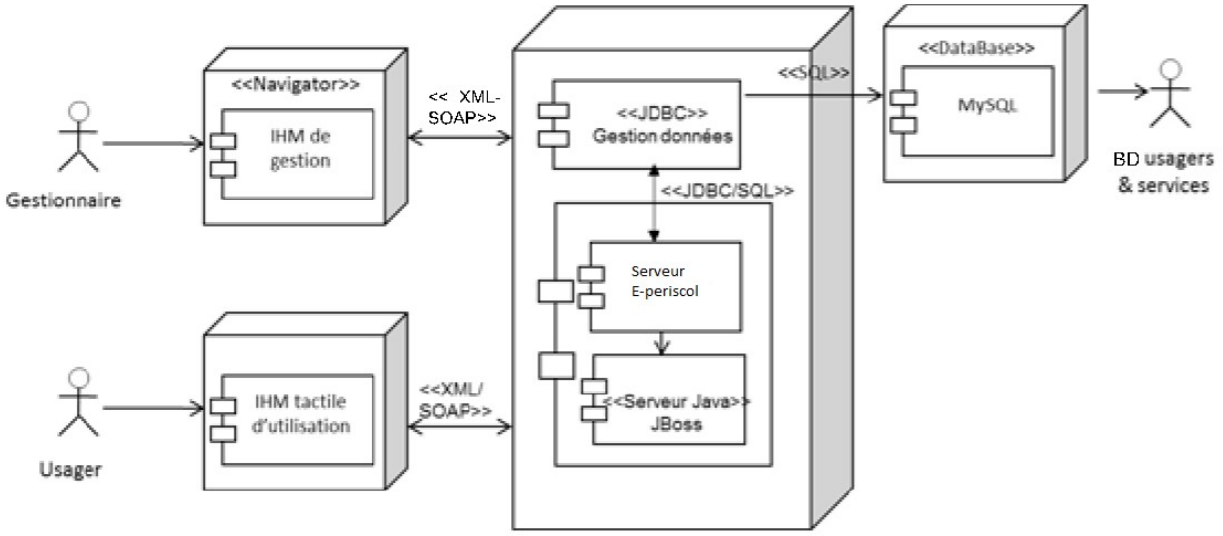






III. Description de l'architecture

1) Diagramme de nœuds



IV. Répartition du travail

	Fonctions à développer et tâches à effectuer
Tous les étudiants	Mise en œuvre et appropriation des environnements de développement. Diagrammes de séquences et de classes de niveau système. Rédaction des spécifications générales.
Étudiant 1 (Dorian Caup)	Installation du serveur Java. Réalisation de la partie serveur des inscriptions, désinscriptions, consultations de service périscolaires et des statistiques Rédacteur de la conception et de l'implémentation de sa partie. *Rédaction de la partie installation du serveur Java et co-rédaction du manuel d'installation de l'application*

Étudiant 2 (Clément Peytureau)	Installation du serveur Java. Réalisation de la partie cliente de la gestion des statistiques sur les services périscolaires Rédacteur de la conception et de l'implémentation de sa partie. *Co-rédaction du manuel d'installation.* *Coordonnateur de la réalisation du dossier technique.*
Étudiant 3 (Baptiste Cahuziere)	Installation de la BD MySQL Usagers & Services. Réalisation du modèle d'information de l'application. Remplissage de la base de données. Réalisation de la partie d'accès aux données des usagers et des services Rédacteur de la conception et de l'implémentation de sa partie. *Rédaction de la partie installation de MySQL* Responsable des points d'avancement du projet.
Étudiant 4 (Romain Gaillard)	Installation du sdk Android IHM tactile d'utilisation des services périscolaires sous Android Rédacteur de la conception et de l'implémentation de sa partie. *Co-rédaction du manuel d'utilisation de l'application.* *Coordonnateur de la réalisation du dossier technique.*
Étudiant 5 (Cédric Cognard)	Installation de l'environnement de développement sous iOS IHM tactile d'utilisation des services périscolaires sous iOS

	Rédacteur de la conception et de l'implémentation de sa partie. *Co-rédaction du manuel d'utilisation de l'application.*
Étudiant 1, 2, 3	Intégration du de l'application serveur Java, du SGBD et du client de gestion des statistiques
Étudiant 1, 3, 4 et 5	Intégration du de l'application serveur Java, du SGBD et des clients tactiles de l'utilisation des services périscolaires
Tous les étudiants	Intégration de l'application *Réalisation du dossier technique du projet.*

V. Description de la partie de Dorian Caup – Serveur Jboss

1) Description Web Service JBoss :

L'objectif de ce serveur est de répondre aux attentes des applications e-PeriScol, et de créer un pont reliant les applications au serveur de données.
Il s'agit plus précisément d'une liste de services nécessaires au fonctionnement des applications e-PeriScol.
Voici une liste non-exaustive, à titre d'exemple, des services proposés :

```
public Connexion SeConnecter(String Email, String Mdp);
public int[] Get_NbrEnfantsInscrit();
public int Get_Pourcentage(int i);
public int CreerCompte (String Email, String Mdp, String Nom, int NbrEnfants,
String AdrPostal);
public boolean AjouterEnfant(String Nom ,String Prenom ,boolean Sexe, String
DateNaiss, ArrayList<String> Activite, String Photo, int ID_Fam);
public boolean AjouterService(String Nom , String a ,int b ,int c,int d);
public boolean InscrireEnfantService(int ID_Enfant, int ID_Service);
public Enfant[] Get_Enfant(int ID_Fam);
public Catalogue Get_All_Service();
public boolean GererCompte(int ID, String Email, String Mdp, int NbrEnfant,
String AdrPostal);
public boolean ModifierEnfant(int ID, String Nom ,String Prenom ,boolean
Sexe ,String Date, short Age,String[] Activite, String Photo);
public boolean SupprimerEnfant(int ID);
public boolean MdpOublie(String Email) throws MessagingException,
NamingException;
```

Les noms des méthodes doivent pouvoir suffire à identifier le service associé.
Retrouvez en annexe le fichier de description du Web service, autrement appelé, fichier WSDL.

2) Mise en œuvre :

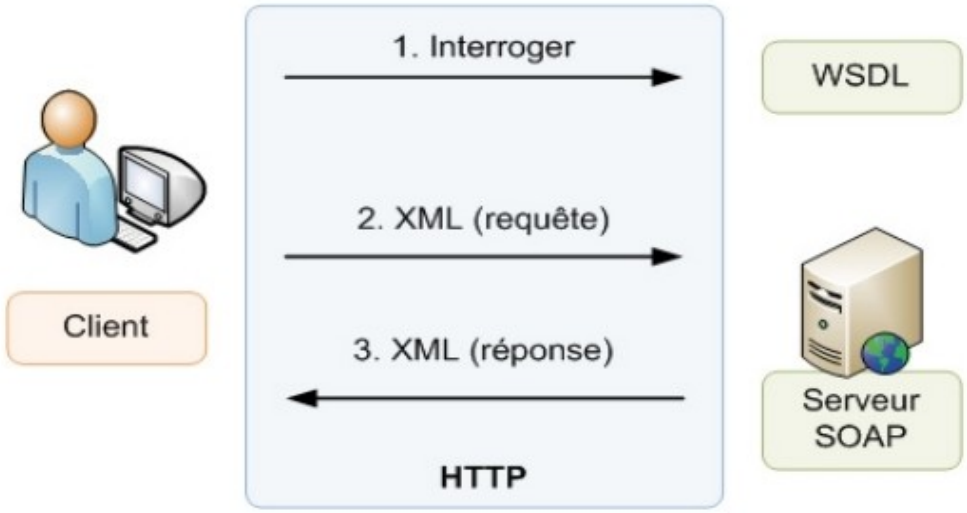
a. Le protocole SOAP :

Le protocole SOAP est le protocole que l'on va utiliser pour faire communiquer les applications nomades, et l'application du gestionnaire avec le serveur, en mode Web Service.
Une des définitions pour ce protocole est la suivante :

> SOAP (ancien acronyme de *Simple Object Access Protocol*) est un protocole de RPC orienté objet bâti sur XML.
>
> Il permet la transmission de messages entre objets distants, ce qui veut dire qu'il autorise un objet à invoquer des méthodes d'objets physiquement situés sur un autre serveur. Le transfert se fait le plus souvent à l'aide du protocole HTTP, mais peut également se faire par un autre protocole, comme SMTP.

La clé de cette définition ce trouve ici : `il autorise un objet à invoquer des méthodes d'objets physiquement situés sur un autre serveur[…]`

Il s'agit d'un protocole de communication permettant notamment à un client d'utiliser des méthodes d'un objet qui se situent sur un serveur, appelées Web Service.
Au niveau du message transmis par ce protocole, que nous appelons enveloppe SOAP, il se présente comme un fichier XML. C'est à dire avec des balises, des conteneurs et des contenus.
Ci-dessous un synoptique général du protocole :



-Le WSDL du « Serveur SOAP » fournit toutes les informations nécessaires pour créer une requête SOAP.
-Un Client utilisant le protocole SOAP peut créer alors une requête vers le « Serveur SOAP» en question.
Dans sa demande est donc spécifié la méthode distante à utiliser.
-Le « Serveur SOAP » répond à cette requête, et formule une réponse SOAP.
-Le Client peut alors travailler grâce au fichier XML reçu, dans lequel il trouve la réponse à sa question.

A retenir :

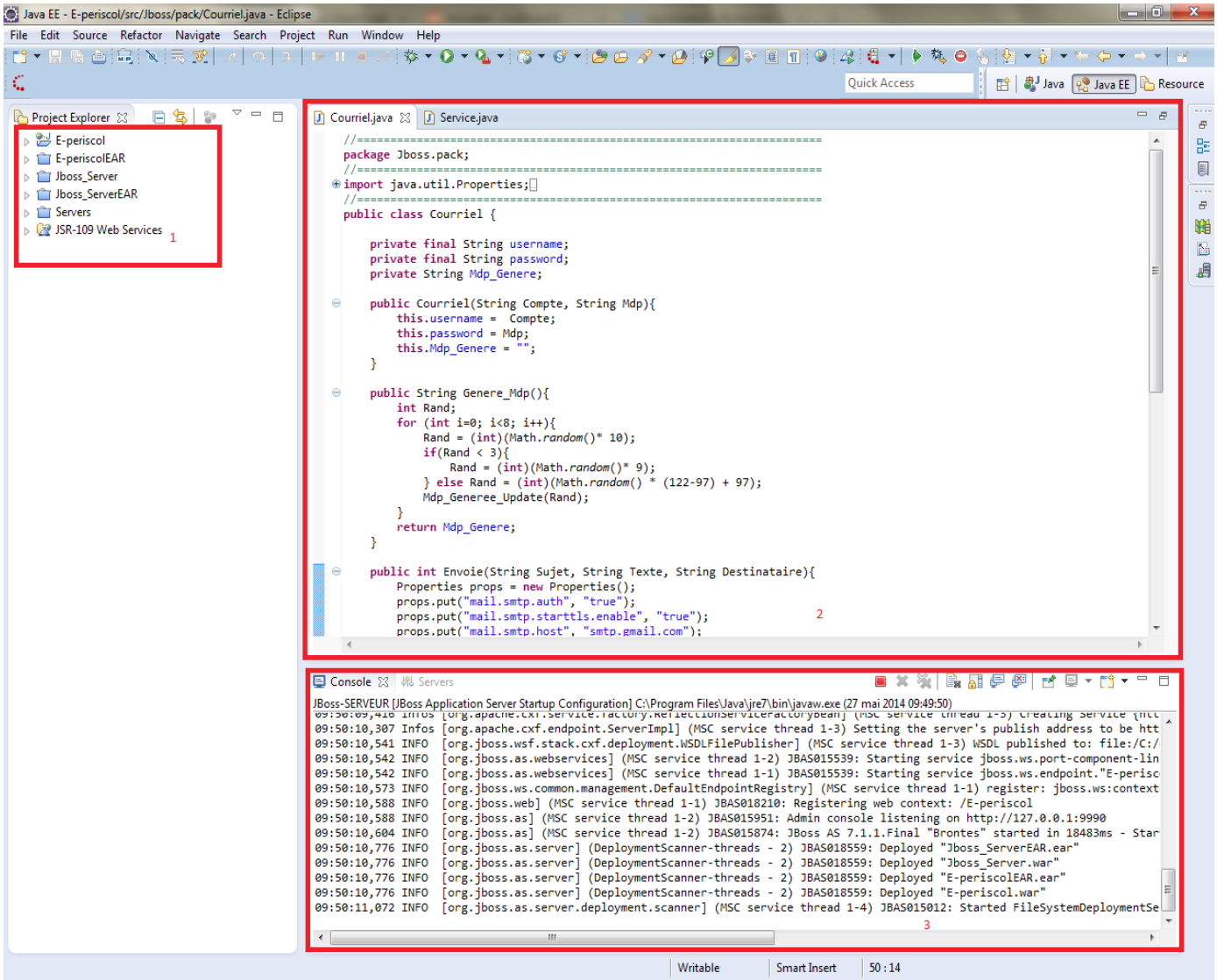
-Le fichier WSDL(Web Service Description Langage) d'un Web Service permet de décrire ce serveur et donc d'implémenter facilement des communication SAOP. Il est donc important de savoir moduler à souhait ce fichier !

-Le protocole SOAP est un échange de fichier sous format XML.

-Il est possible de transférer des objets via le protocole SOAP, et même des objets « complexes » que vous pouvez créer vous même.

b) Environnement de travail :

Nous travaillons sur Eclipse Juno pour Java EE :


Java EE - E-periscol/src/Jboss/pack/Courriel.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Java EE
Resource
Project Explorer
E-periscol
E-periscolEAR
Jboss_Server
Jboss_ServerEAR
Servers
JSR-109 Web Services
1
Courriel.java
Service.java
package Jboss.pack;
import java.util.Properties;
public class Courriel {
private final String username;
private final String password;
private String Mdp_Genere;
public Courriel(String Compte, String Mdp){
this.username = Compte;
this.password = Mdp;
this.Mdp_Genere = "";
}
public String Genere_Mdp(){
int Rand;
for (int i=0; i<8; i++){
Rand = (int)(Math.random()* 10);
if(Rand < 3){
Rand = (int)(Math.random()* 9);
} else Rand = (int)(Math.random() * (122-97) + 97);
Mdp_Generee_Update(Rand);
}
return Mdp_Genere;
}
public int Envoie(String Sujet, String Texte, String Destinataire){
Properties props = new Properties();
props.put("mail.smtp.auth", "true");
props.put("mail.smtp.starttls.enable", "true");
props.put("mail.smtp.host", "smtp.gmail.com");
2
Console
Servers
JBoss-SERVEUR [JBoss Application Server Startup Configuration] C:\Program Files\Java\jre7\bin\javaw.exe (27 mai 2014 09:49:50)
09:50:10,307 Infos [org.apache.cxf.endpoint.ServerImpl] (MSC service thread 1-3) Setting the server's publish address to be htt
09:50:10,541 INFO [org.jboss.wsf.stack.cxf.deployment.WSDLFilePublisher] (MSC service thread 1-3) WSDL published to: file:/C:/
09:50:10,542 INFO [org.jboss.as.webservices] (MSC service thread 1-2) JBAS015539: Starting service jboss.ws.port-component-lin
09:50:10,542 INFO [org.jboss.as.webservices] (MSC service thread 1-1) JBAS015539: Starting service jboss.ws.endpoint."E-perisc
09:50:10,573 INFO [org.jboss.ws.common.management.DefaultEndpointRegistry] (MSC service thread 1-1) register: jboss.ws:context
09:50:10,588 INFO [org.jboss.web] (MSC service thread 1-1) JBAS018210: Registering web context: /E-periscol
09:50:10,588 INFO [org.jboss.as] (MSC service thread 1-2) JBAS015951: Admin console listening on http://127.0.0.1:9990
09:50:10,604 INFO [org.jboss.as] (MSC service thread 1-2) JBAS015874: JBoss AS 7.1.1.Final "Brontes" started in 18483ms - Star
09:50:10,776 INFO [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS018559: Deployed "Jboss_ServerEAR.ear"
09:50:10,776 INFO [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS018559: Deployed "Jboss_Server.war"
09:50:10,776 INFO [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS018559: Deployed "E-periscolEAR.ear"
09:50:10,776 INFO [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS018559: Deployed "E-periscol.war"
09:50:11,072 INFO [org.jboss.as.server.deployment.scanner] (MSC service thread 1-4) JBAS015012: Started FileSystemDeploymentSe
3
Writable
Smart Insert
50 : 14

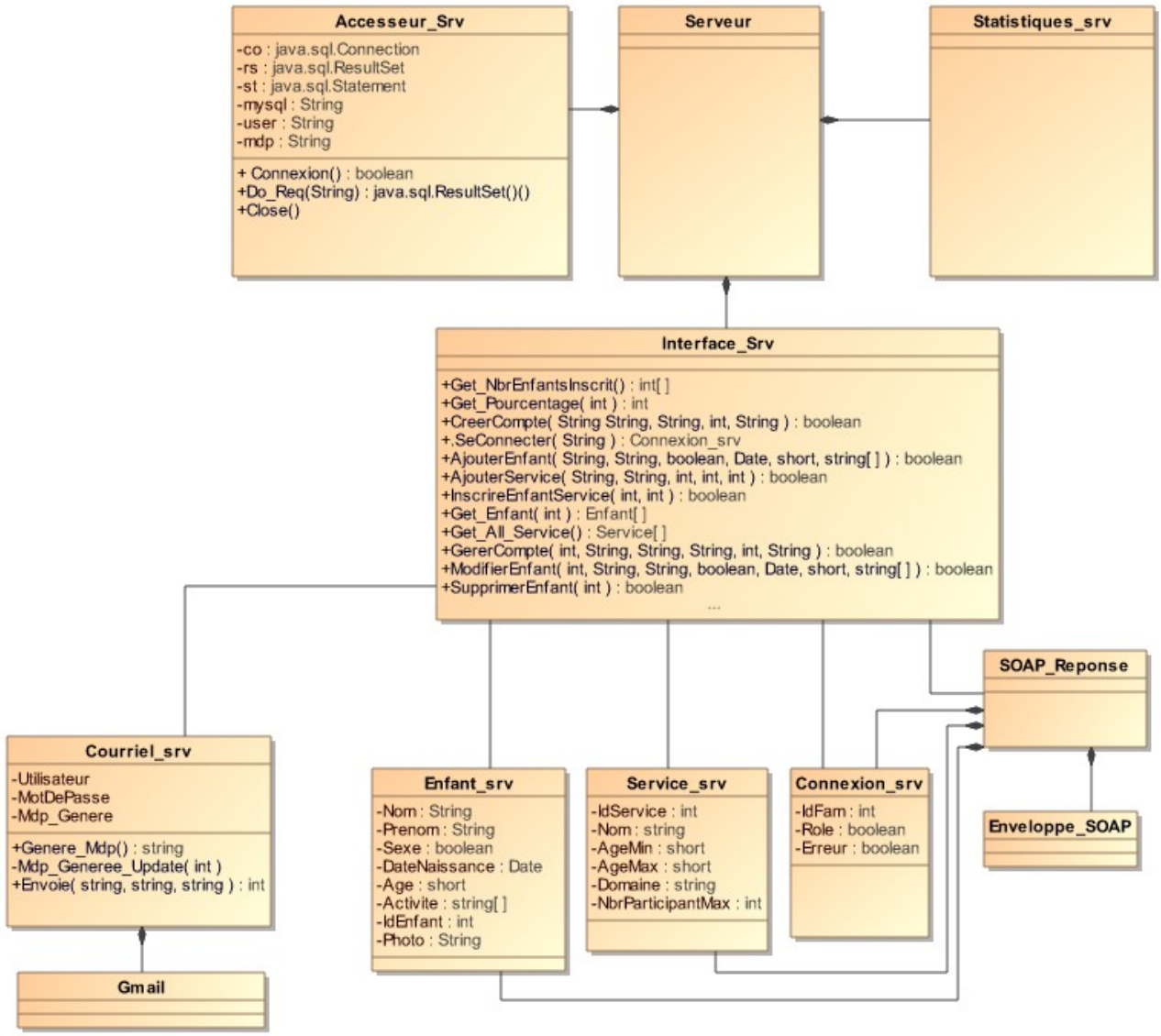
1 : Exploration des projet.

2 : Code source.

3 : Console du serveur, et gestion du serveur et de son déployment.

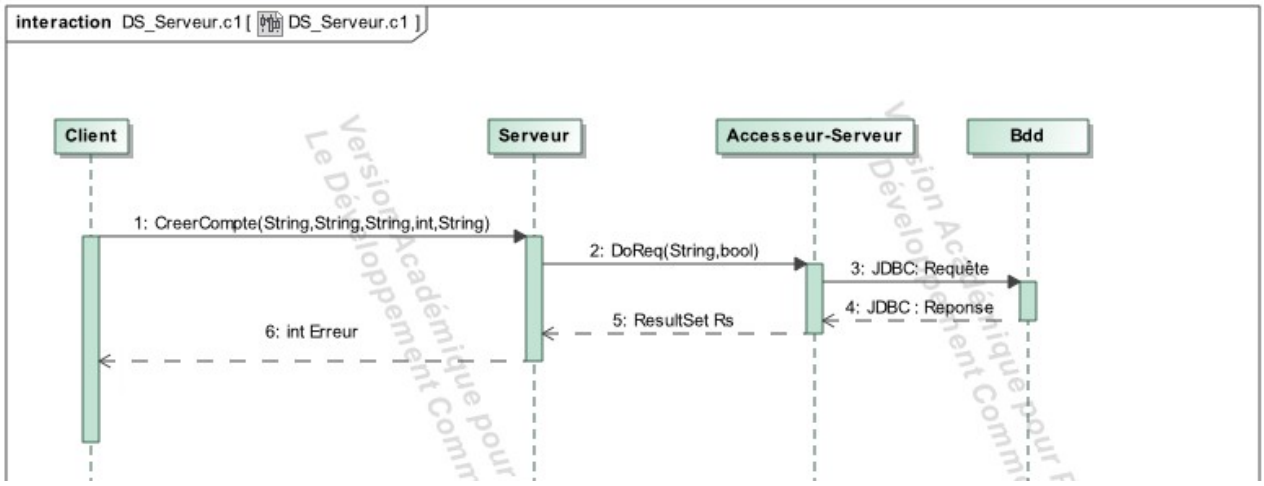
2) Conception détaillée:

a) Diagramme de classe détaillé :



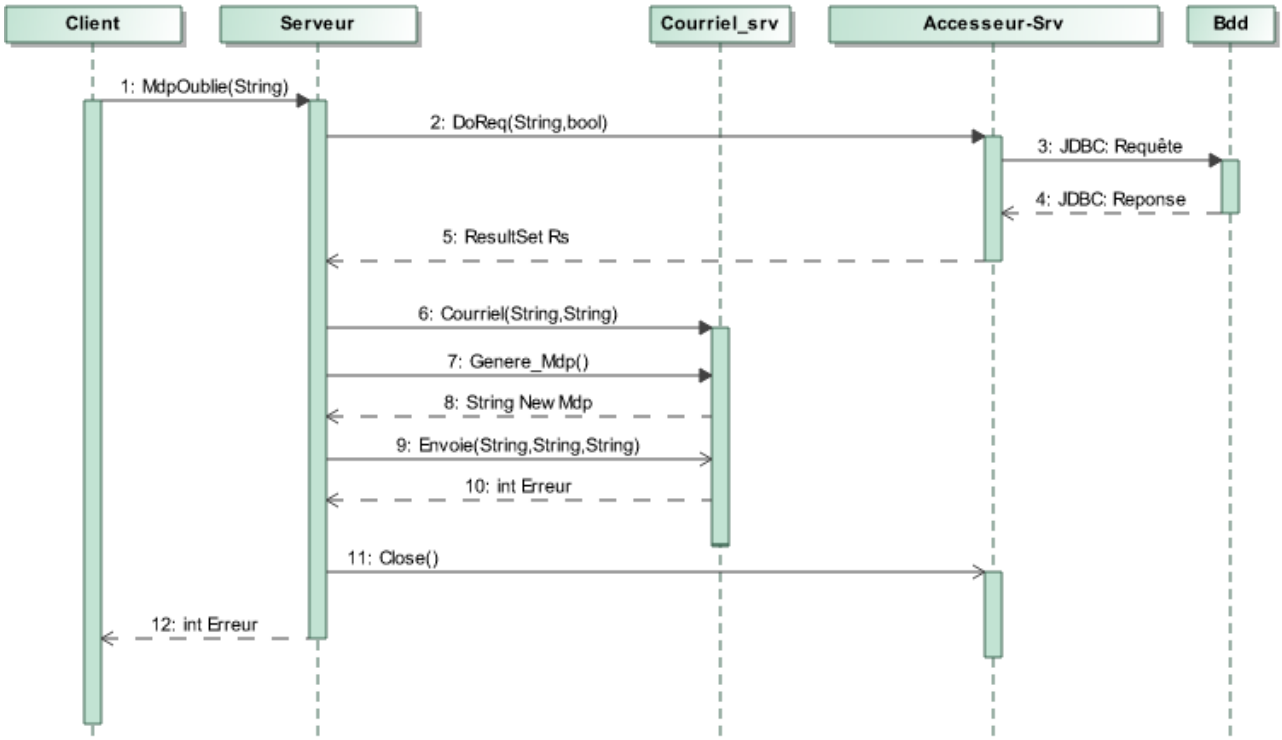
http://image.noelshack.com/fichiers/2014/22/1401539861-dc-serveur.png

b) Diagramme de séquence détaillé : CreerCompte() :



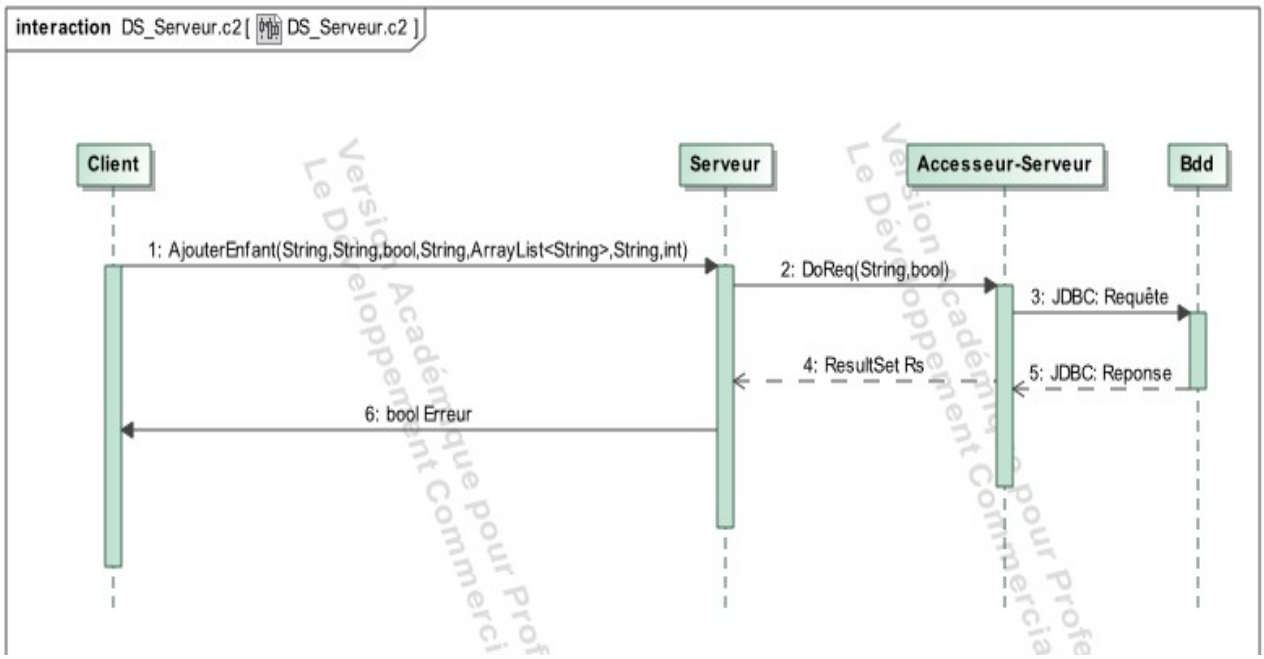
http://www.noelshack.com/2014-22-1401185142-ds-serveur-c1.png

c) Diagramme de séquence détaillé : MdpOublie(String) :



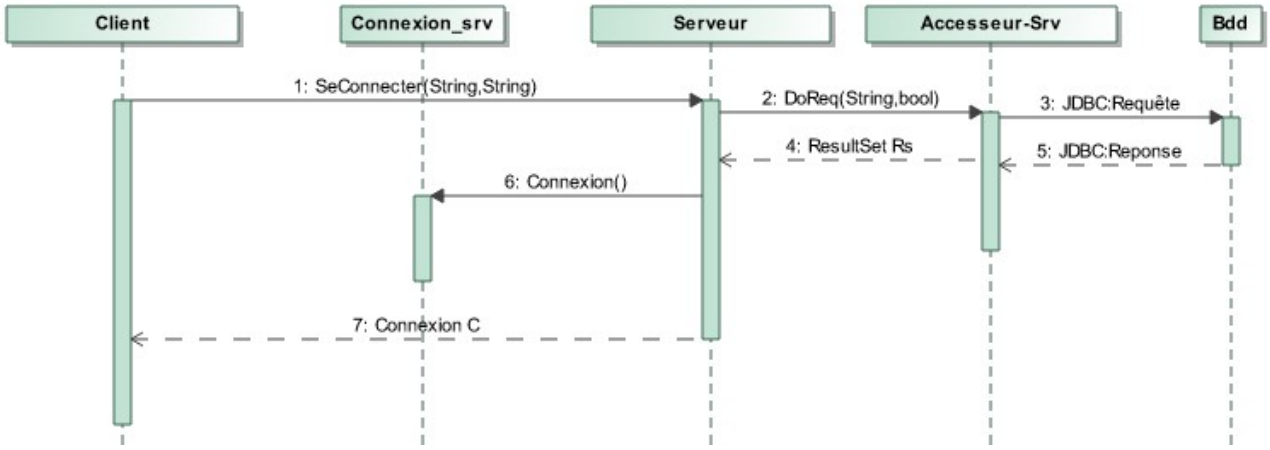
http://image.noelshack.com/fichiers/2014/22/1401540081-ds-srv-mdp.png

d) Diagramme de séquence détaillé AjouterEnfant(String, String, bool, String, ArrayList<String>, String, int) :



http://image.noelshack.com/fichiers/2014/22/1401185144-ds-serveur-c2.png

e) Diagramme de séquence détaillé SeConnecter(String,String) :



http://image.noelshack.com/fichiers/2014/22/1401540229-ds-srv-connexion.png

3) Implémentation :

Dans cette partie seront présentées quelques méthodes du Web Service qui ont des particularités, ou un algorithme particulier.

a) Notions clés sur la technologie utilisée

a.1. Les annotations JAX WS :

Jax WS est une API « application programming interface » de Java utilisé pour les Web Services. Elle permet via des annotations de moduler le fichier WSDL d'un Web Service à souhait.

```
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
```

Voici donc les 4 annotations utilisée :

```
@WebService(targetNamespace="http://192.168.3.5:8080/Actions_Client")
public class Actions_Client implements Client {
…
}
```

Cette annotation permet de spécifier que la classe suivante est un Web Service.

```
@WebMethod(action = "http://D306-P01:8080/e-
PeriScol/Actions_Client",operationName ="SeConnecter")
```

Cette annotation permet de spécifier que la méthode qui va suivre est une méthode du Web Service.

```
public Connexion SeConnecter(@WebParam(name = "Email")String Email,
@WebParam(name = "Mdp")String Mdp){
```

Cette annotation permet de spécifier les paramètres qui devront être envoyer par l'utilisateur du Web Service lors de l'appel de cette méthode.

```
@WebResult(name = "Catalogue", partName = "Catalogue"
```

Cette annotation permet de spécifier le retour de la méthode afin d'aider l'utilisateur à l'identifier. Le WSDL lui est un fichier de description du Web Service, il permet donc à l'utilisateur du Web Service de comprendre comment est composé celui-ci.

```
<wsdl:definitions name="Actions_ClientService"
targetNamespace="http://192.168.3.5:8080/Actions_Client"><wsdl:types><xs:schema
attributeFormDefault="unqualified" elementFormDefault="unqualified"
targetNamespace="http://192.168.3.5:8080/Actions_Client"><xs:element
name="AjouterEnfant" type="tns:AjouterEnfant"/><xs:element
name="AjouterEnfantResponse" type="tns:AjouterEnfantResponse"/><xs:element
name="AjouterService" type="tns:AjouterService"/><xs:element
name="AjouterServiceResponse" type="tns:AjouterServiceResponse"/><xs:element
name="CreerCompte" type="tns:CreerCompte"/><xs:element
name="CreerCompteResponse" type="tns:CreerCompteResponse"/>
...
```

On peut apercevoir d'une part la liste des méthodes du WebService disponible.

```
<xs:complexType name="AjouterEnfant"><xs:sequence><xs:element minOccurs="0"
name="Nom" type="xs:string"/><xs:element minOccurs="0" name="Prenom"
type="xs:string"/><xs:element name="Sexe" type="xs:boolean"/><xs:element
minOccurs="0" name="DateNaiss" type="xs:string"/><xs:element
maxOccurs="unbounded" minOccurs="0" name="Activite"
type="xs:string"/><xs:element minOccurs="0" name="Photo"
type="xs:string"/><xs:element name="ID_Fam" type="xs:int"/></xs:sequence>
</xs:complexType>
```

Et d'autre part les paramètres de chaque méthodes ainsi que leur valeur de retour.

```
<xs:complexType name="AjouterEnfantResponse"><xs:sequence><xs:element
name="return" type="xs:boolean"/></xs:sequence>
</xs:complexType>
```

a.2. javax.xml.bind.annotation.XmlType :

Il est parfois nécessaire de retourner des objets complexe par protocole SOAP. Pour cela, il faut donner à l'utilisateur la composition de l'objet en question.

```
package Jboss.pack;
//======================================================================
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;
//======================================================================
@XmlType(name = "Service")
public class Service {
      @XmlElement(name="Nom")
      private String Nom;
      @XmlElement(name="Domaine")
      private String Domaine;
      @XmlElement(name="AgeMin")
      private short AgeMin;
      @XmlElement(name="AgeMax")
      private short AgeMax;
      @XmlElement(name="NbrMax")
      private int NbrMax;
      public Service(){ … }
```

On utilise alors les annotations XML de javax. A chaque element de ma classe, qui va permettre d'instancier mes objets, j'assigne une une annotation spécifiant le nom qu'aura cette element dans le WSDL. Autrement dit, il s'agit d'une description de l'objet complexe. Voici donc un exemple concret :

```
<xs:complexType name="Get_All_ServiceResponse">
<xs:sequence><xs:element minOccurs="0" name="Catalogue" type="tns:Catalogue"/>
</xs:sequence>
```

On voit ici qu'on utilise un objet complexe Catalogue, qui est en fait un tableau d'objet complexe Service.

```
</xs:complexType><xs:complexType name="Catalogue">
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="Service"
type="tns:Service"/>
</xs:sequence>
</xs:complexType>
```

On décrit alors ici l'objet complexe Service en question.

```
<xs:complexType name="Service">
<xs:sequence>
<xs:element minOccurs="0" name="Nom" type="xs:string"/>
<xs:element minOccurs="0" name="Domaine" type="xs:string"/>
<xs:element name="AgeMin" type="xs:short"/>
<xs:element name="AgeMax" type="xs:short"/><xs:element name="NbrMax"
type="xs:int"/>
</xs:sequence>
</xs:complexType>
```

a.3. JDBC : Java Data Base Connector :

Ce protocole permet la communication entre une application Java et une base de donnée, via la création de requêtes SQL.

Pour ce faire, nous avons créer la classe ci-après :

```
public class Actions_Bdd {
	private String mysql,user,mdp;
	private Connection co = null;
	private ResultSet rs = null;
	private Statement st = null;
	public Actions_Bdd(String a, String b, String c){
		mysql = a;
		user = b;
		mdp = c;
	}
	public boolean Connexion(){
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {e.printStackTrace();}
		try {
			co = DriverManager.getConnection(mysql,user,mdp);
			st = co.createStatement();
			return true;
		} catch (SQLException e) {System.out.println("Connection
Failed!\n");e.printStackTrace();}
		return false;
	}
	public ResultSet Do_Req(String a, boolean b){
	   if(b==false){
		try {
				rs = st.executeQuery(a);
```

```
                if (rs != null)return rs;
            } catch (SQLException e) { e.printStackTrace(); }
         } else
                try { st.executeUpdate(a); } catch (SQLException e)
{ e.printStackTrace(); }
          return null;
       }
public void Close() {
       try {
           this.st.close();
           this.co.close();
           this.co.close();
       } catch (SQLException e) {
         e.printStackTrace();
       }
   }
}
```

Le constructeur initialise les données permettant la connexion à la base de donnée en question.

-La méthode Connexion() établit la connexion avec la base de donnée.

-La méthode Do_Req(String,boolean) permet d'envoyer une requête sur le base de donnée en spécifiant si il s'agit d'une requête de Lecture ou D'écriture sur celle-ci.

-La méthode Close() permet de fermer proprement la connexion à la base de donnée.

La classe est simple à implémenter il suffit de la rajouter au package du Web Service et d'instancier un objet Accesseur_srv.

a.4. Petit service supplémentaire :

Une des éventualité lorsque l'on gère un serveur est de pouvoir communiquer par e-mail avec les utilisateurs du Web Service. Pour cela, la classe suivante à été mise en place :

```
public class Courriel {

      private final String username;
      private final String password;
      private String Mdp_Genere;

      public Courriel(String Compte, String Mdp){
            this.username =  Compte;
            this.password = Mdp;
            this.Mdp_Genere = "";
      }

      public String Genere_Mdp(){
            int Rand;
            for (int i=0; i<8; i++){
                  Rand = (int)(Math.random()* 10);
                  if(Rand < 3){
                        Rand = (int)(Math.random()* 9);
                  } else Rand = (int)(Math.random() * (122-97) + 97);
                  Mdp_Generee_Update(Rand);
            }
            return Mdp_Genere;
      }
```

```
	public int Envoie(String Sujet, String Texte, String Destinataire){
		Properties props = new Properties();
		props.put("mail.smtp.auth", "true");
		props.put("mail.smtp.starttls.enable", "true");
		props.put("mail.smtp.host", "smtp.gmail.com");
		props.put("mail.smtp.port", "587");

		Session session = Session.getInstance(props,
		  new javax.mail.Authenticator() {
			protected PasswordAuthentication getPasswordAuthentication()
{
				return new PasswordAuthentication(username, password);
			}
		  });

		try {
			Message message = new MimeMessage(session);
			message.setFrom(new InternetAddress("noreply-
eperiscol@gmail.com"));


message.setRecipients(Message.RecipientType.TO,InternetAddress.parse(Destinatai
re));
			message.setSubject(Sujet);
			message.setText(Texte);
			Transport.send(message);
			System.out.println("Mail envoyer a : "+Destinataire);
		} catch (MessagingException e) { return 2;}
		return 0;
	}
	//========================================================================
	private void Mdp_Generee_Update(int Rand){
		StringBuilder Sb = new StringBuilder();
		if(Rand >= 0 && Rand < 10){
			this.Mdp_Genere = this.Mdp_Genere+(Integer.toString(Rand));
		} else {
			Sb.append((char)Rand);
			this.Mdp_Genere = this.Mdp_Genere+Sb.toString();
		}
	}
}
```

Le constructeur permet d'initialiser principalement l'e-mail de l'administrateur.

-La méthode Genere_Mdp() permet de générer un mot de passe aléatoire de 8 chars.
-La méthode Envoie(String,String,String) permet d'envoyer un message à un destinataire.

b) Étude du code Mdp_oublie(String) :

```
@WebMethod(action = "http://D306-P01:8080/e-
PeriScol/Actions_Client",operationName ="MdpOublie")
@WebResult(name = "Error", partName = "Error")
public int MdpOublie(@WebParam(name = "Email")String Email) throws
MessagingException, NamingException {
	bddA = new Actions_Bdd(this.mysql, this.user, this.mdp);
	int Erreur = 0;
	if (bddA.Connexion()){
		Rs = null;
```

```
            boolean Continue = false;
            Rs = bddA.Do_Req("SELECT ID FROM compte WHERE AdrMail =
'"+Email+"'",                false);
            try { Continue = Rs.first(); } catch (SQLException e1)
                  { e1.printStackTrace(); }
            if (Rs != null && Continue != false ){
                  Courriel Mail = new
Courriel("epericol@gmail.com","mdpadmin");
                  String NewMdp = Mail.Genere_Mdp();
                  String Titre = "e-PeriScol- Nouveau mot de passe";
                  String Corps = "Bonjour, \n Suite à votre demande de
                        récupération de mot de passe, nous vous transmettons "+
                  "votre nouveau mot de passe : "+NewMdp+"\n"+ "Vous pouvez dès
                        à présent vous connecter avec celui-ci sur
l'application E-                              periscol."+"\nPour information, il
vous est possible de le                             changer dans les
paramètres de votre compte.\n\n A bientôt,                        l'équipe e-
PeriScol";
                  Erreur = Mail.Envoie(Titre,Corps,Email);
                  bddA.Do_Req ("UPDATE compte SET Mdp = password('"+NewMdp+"')
                        WHERE AdrMail = '"+Email+"' ", true);
                  bddA.Close();
            } else return 1;
      }
      return Erreur;
}
```

Il s'agit ici de la méthode Mdp_Oublie(String) qui permet d'envoyer un message sur la boite mail de la personne ayant oublié le sien. Un nouveau mot de passe est généré automatiquement, mot de 8 char, et un message est délivré à la personne concernée.

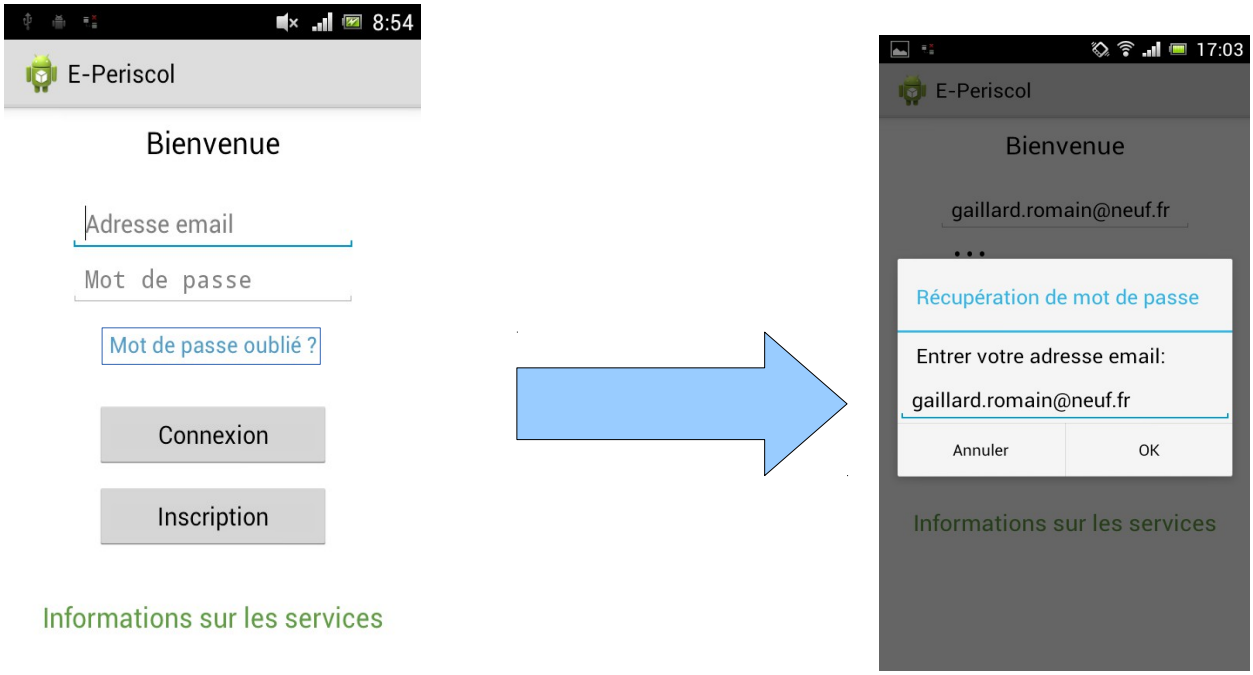
c) Étude du code Get_All_Service(String) :

```
@WebMethod(action = "http://D306-P01:8080/e-
PeriScol/Actions_Client",operationName ="Get_All_Service")
      @WebResult(name = "Catalogue", partName = "Catalogue")
      public Catalogue Get_All_Service() {
            bddA = new Actions_Bdd(this.mysql, this.user, this.mdp);;
            Service S;
            Catalogue Cata = null;
            Service[] S2 = null;
            int IDMax = 0;
            if (bddA.Connexion()){
                  Rs = bddA.Do_Req("SELECT ID_Service FROM service ORDER BY
ID_Service                DESC LIMIT 1",false);
                  try {
                        Rs.first();
                        IDMax = Rs.getInt(1);
                        Cata = new Catalogue(IDMax);
                        S2 = new Service[IDMax];
                        for (int i=1; i<(IDMax+1); i++){
                              Rs2 = bddA.Do_Req("SELECT * FROM service WHERE
                              ID_Service = "+i+" ", false);
                              try{
                                    Rs2.first();
                                    S = new Service();
                                    S.Set_Nom(Rs2.getString(2));
                                    S.Set_Domaine(Rs2.getString(3));
                                    S.Set_AgeMin((short)Rs2.getInt(4));
```

```
                                    S.Set_AgeMax((short)Rs2.getInt(5));
                                    S.Set_NbrMax(Rs2.getInt(6));
                                    S2[i-1] = S;
                                } catch (SQLException e) { e.printStackTrace();}
                            }
                            Cata.Set_Catalogue(IDMax, S2);
                        } catch (SQLException e) { e.printStackTrace(); }
                }
                return Cata;
        }
```

4) Test de validation :

Nous allons suivre le cas d'utilisation « Mdp_Oublie(String) », pour cela, nous devons en premier nous situer au niveau de l'application Android.



Une fois le bouton OK pressé la méthode du WebService Mdp_Oublie(String) est appelée avec l'e-mail en question. Le protocole SOAP entre alors en action, voilà l'échange effectué :

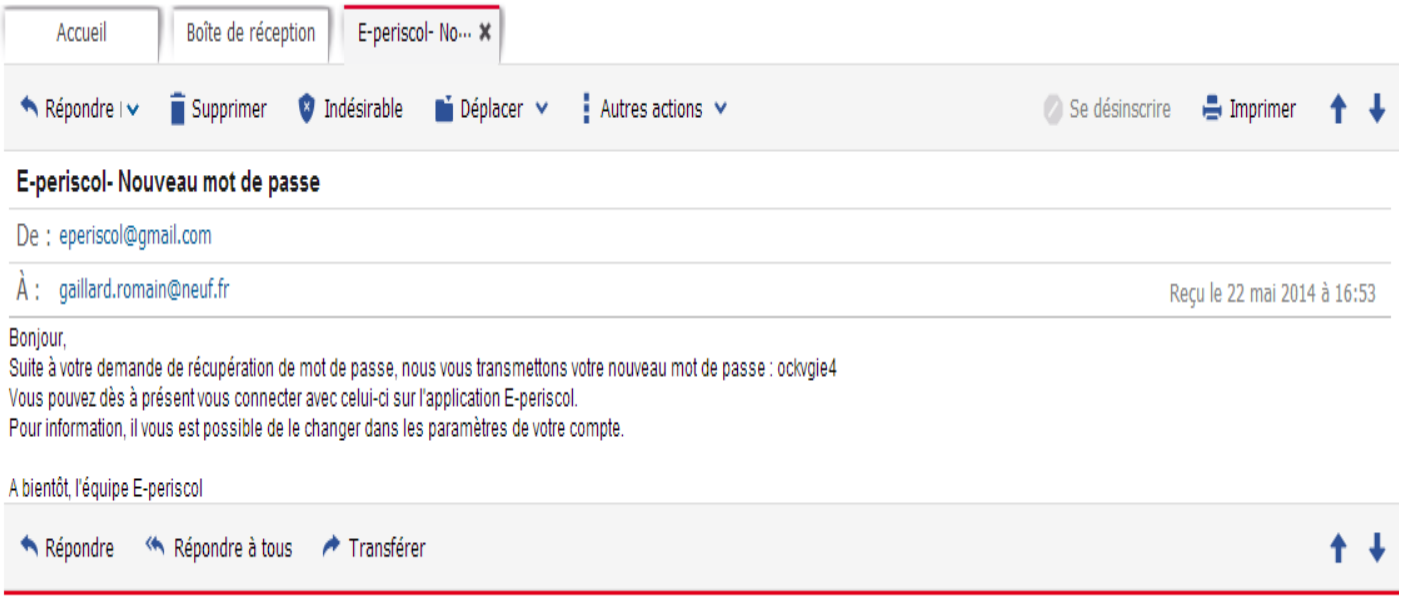
SOAP Requête :

```
<v:Envelope
    xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:d="http://www.w3.org/2001/XMLSchema"
    xmlns:c="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:v="http://schemas.xmlsoap.org/soap/envelope/">
    <v:Header/>
  <v:Body>
    <n0:MdpOublie
        id="o0"
        c:root="1"
        xmlns:n0="http://192.168.3.5:8080/Actions_Client">
      <Email
          i:type="d:string">
          gaillard.romain@neuf.fr
          </Email>
        </n0:MdpOublie>
      </v:Body>
    </v:Envelope>
```

SOAP Réponse

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:MdpOublieResponse
        xmlns:ns2="http://192.168.3.5:8080/Actions_Client">
      <Error>
          0
          </Error>
        </ns2:MdpOublieResponse>
      </soap:Body>
    </soap:Envelope>
```

L'erreur est a 0, donc tout c'est correctement déroulé. On peut terminer ce test en vérifiant d'une part que le mail est envoyé, puis d'autre part que le nouveau mot de passe est effectif.
Remarque: Nous avons créer une adresse email dédiée au compte e-periscol : eperiscol@gmail.com


Accueil
Boîte de réception
E-periscol- No...
Répondre
Supprimer
Indésirable
Déplacer
Autres actions
Se désinscrire
Imprimer
E-periscol- Nouveau mot de passe
De : eperiscol@gmail.com
À : gaillard.romain@neuf.fr
Reçu le 22 mai 2014 à 16:53
Bonjour,
Suite à votre demande de récupération de mot de passe, nous vous transmettons votre nouveau mot de passe : ockvgie4
Vous pouvez dès à présent vous connecter avec celui-ci sur l'application E-periscol.
Pour information, il vous est possible de le changer dans les paramètres de votre compte.
A bientôt, l'équipe E-periscol
Répondre
Répondre à tous
Transférer

5) Bilan personnel :

Pour ma part, j'ai pu travailler sur des concepts et technologies riches me permettant ainsi d'accumuler plusieurs compétences simultanément. J'ai pu découvrir ~~de~~ un nouveau protocole et approfondir des protocoles déjà connus. Créer un Web Service, le gérer puis l'optimiser sont des notions très intéressantes, et permettent d'apprendre à travailler en équipe, à communiquer.
J'ai pu aussi approfondir mes connaissances en Java, et en modélisation UML.

VI. Description de la partie Clément Peytureau - Gestionnaire

1) Description de la partie personnelle

La création d'une IHM graphique en Java permettant la gestion des inscriptions des enfants à une activité périscolaire. Cette application donne au gestionnaire la possibilité de calculer et consulter les statistiques. Le gestionnaire pourra aussi créer, ajouter, et supprimer une activité périscolaire grâce a ses droits administrateur, de l'application.

2) Mise en œuvre

a. Fonctionnement de l'application

Le gestionnaire va pouvoir entrer son identifiant administrateur, c'est à dire qu'il a plus de droits qu'un usager normal, et donc va se connecter à l'application. Cette connexion se fait à l'aide d'un serveur applicatif et d'une base de données. Lors de la connexion, une requête est envoyée au serveur, ce dernier va vérifier si les identifiants correspondent avec ceux présents sur la base de données. L'application tourne autour de ce serveur, sans lui aucune connexion n'est possible. Après sa connexion le gestionnaire peut ensuite, consulter les statistiques, inscrire un enfant à une activité extra-scolaire, rechercher si un enfant est inscrit à une activité…
L'usager va utiliser cette application pour inscrire un ou plusieurs de ses enfants a un activité périscolaire à l'aide d'un smartphone ou bien en sollicitant l'aide du gestionnaire présent a la mairie.

b. Comment mettre en œuvre ?

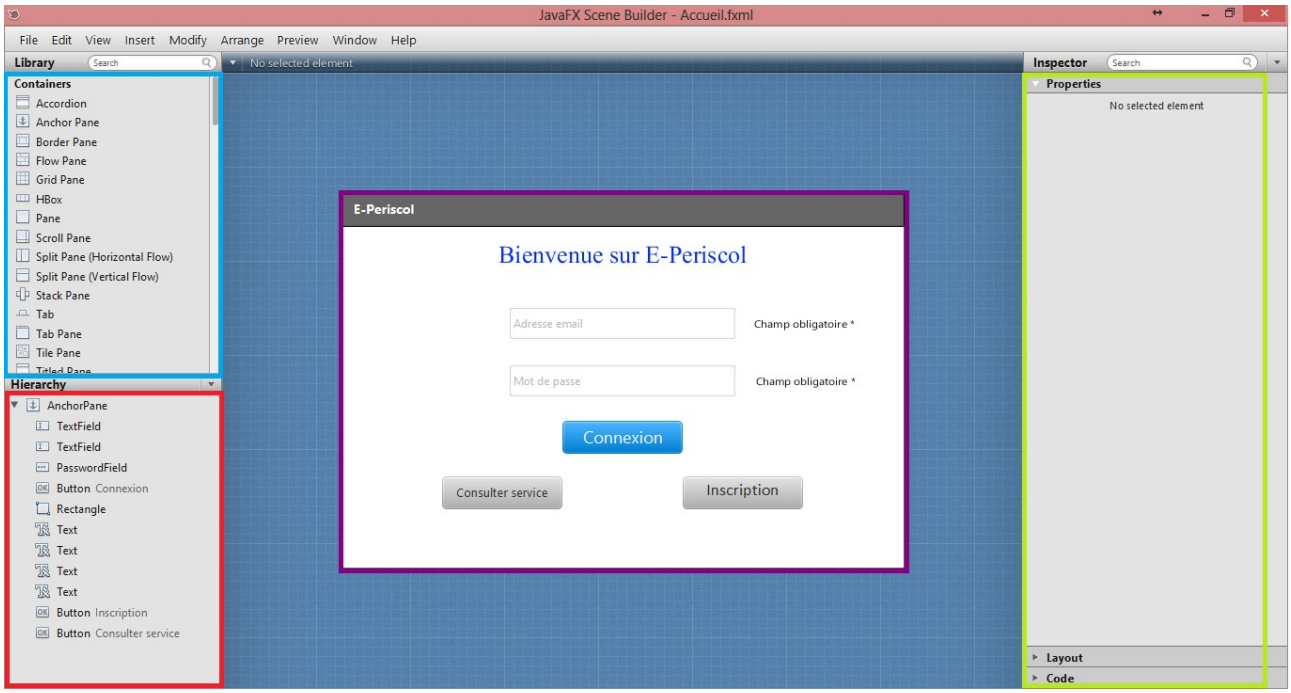
Afin de pouvoir créer cette application nous devons installer le logiciel Eclipse, qui nous permet de créer des projets codé en Java, ainsi que de l'extension JavaFX22.

b.1. Installation de JavaFX22

Pour utiliser JavaFX2, il nous faut tout d'abord installer, « Java Development Kit 7 » incluant la

version 2.2 de JavaFX2, Le Plug-in e(fx)clipse, ainsi qu'au minimum la version 1.1 du SceneBuilder de JavaFX2. Les interfaces utilisées dans JavaFX2 sont construites par le biais du SceneBuilder par un système de drag&drop, il fait ensuite lui-même la transition vers un fichier FXML dont nous aurons ensuite l'utilité pour la création des actions sur notre interface.

b.2. JavaFX2 SceneBuilder



La "**library**" permet la création d'interface grâce à divers catégories d'items:

-Containers : Composants permettant de contenir plusieurs composants se situant dans la partie Controls.

-Controls : Composants utilisés d'une utilisation classique de notre application: bouton,scroll bar, hyperlien.

-Menu : Permet la création d'une barre de menu et sous menu.

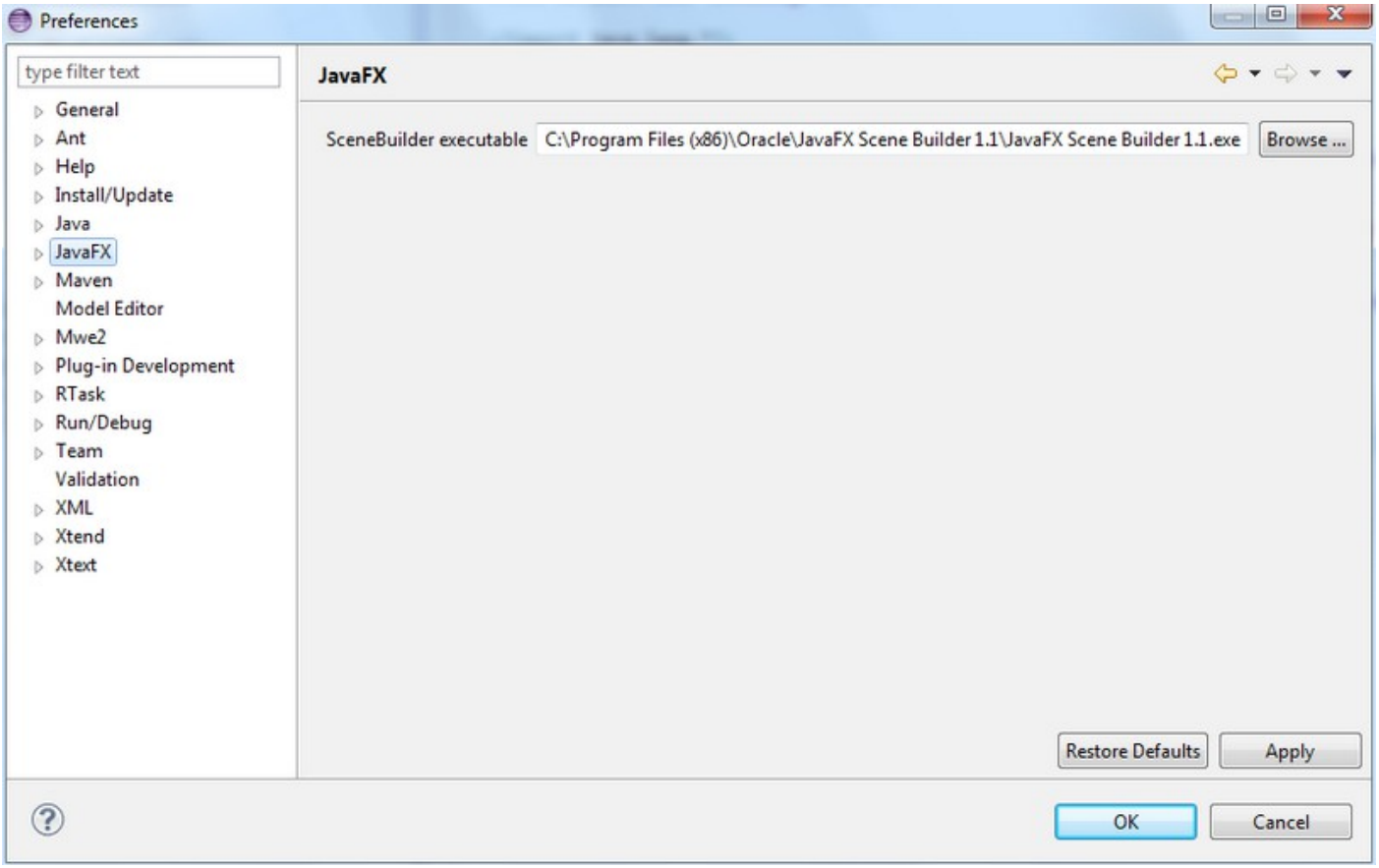
La section "**Hierarchy**" nous indique les items utilisés dans notre SceneBuilder.

L'onglet "**Properties**" sert à changer les propriétés d'un item sélectionné, tels que sa taille, son alignement, son texte, sa police, etc...

C'est la **Scène**, un aperçu de notre IHM.

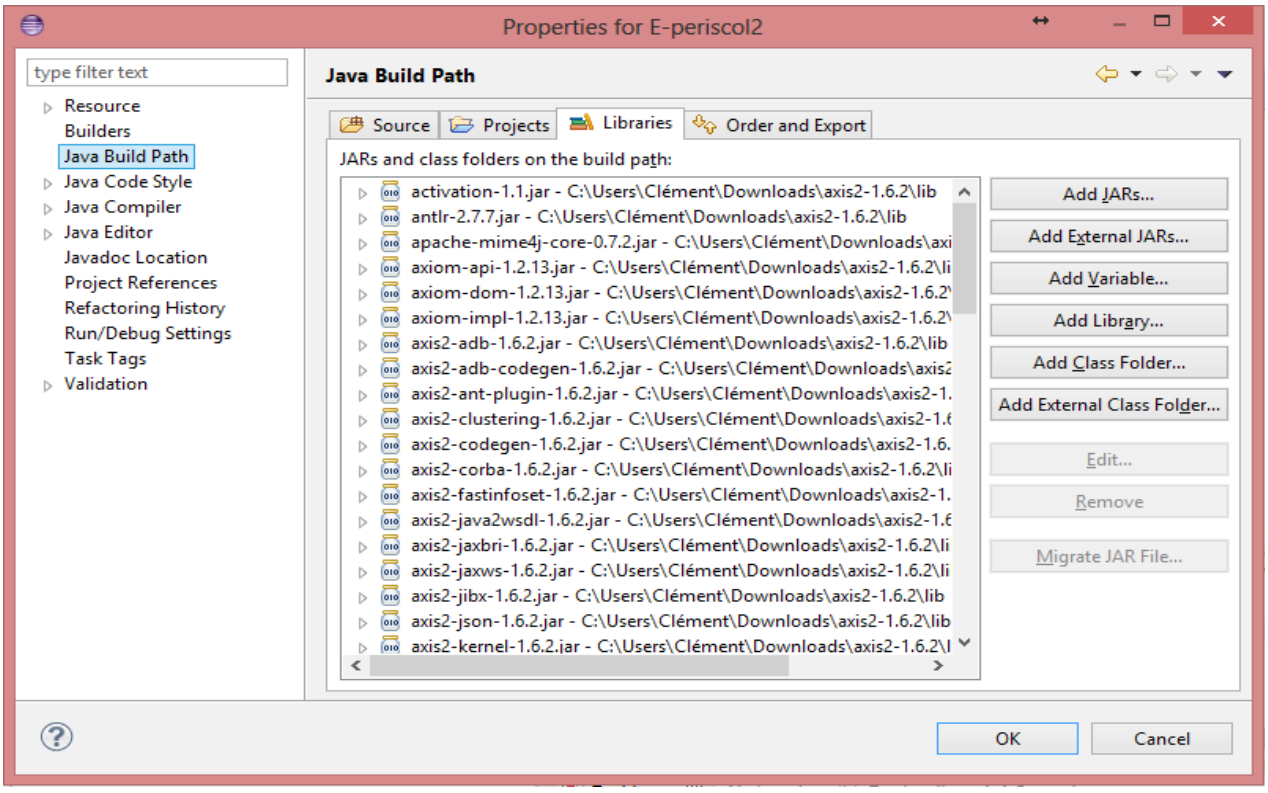
b.3. Intégration SceneBuilder à Eclipse

Une fois le SceneBuilder de JavaFX2 prit en main, nous pouvons ensuite donner le chemin d'installation à Eclipse pour ouvrir les fichiers XML nous permettant de créer nos interfaces.



b.4. Intégration de la librairie Axis2 1.6.2

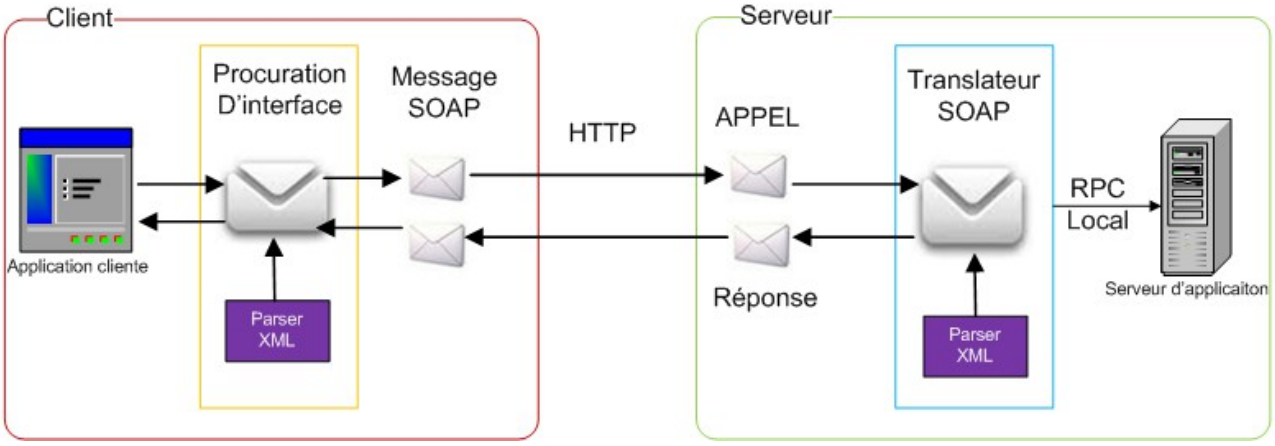
Pour pouvoir intégrer cette librairie il faut d'abord la télécharger sur le site http://axis.apache.org/axis2/java/core/download.cgi. Après avoir récupéré la librairie, je l'intègre à mon projet sous Éclipse.



La librairie Axis2 1.6.2 intègre le protocole SOAP (Simple Object Access Protocol) qui va nous permettre de communiquer et d'échanger des données avec le web service.

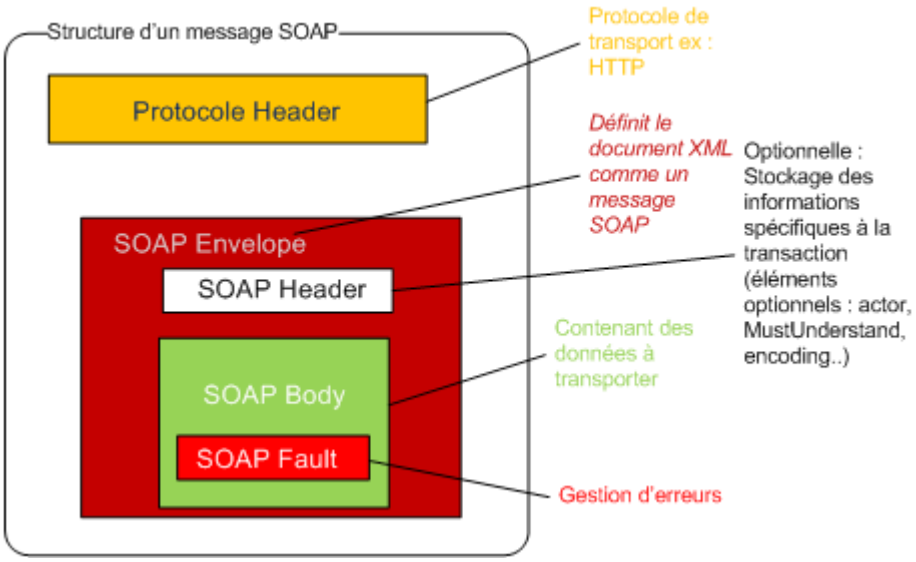
b.5. Fonctionnement du protocole SOAP

SOAP est un protocole d'invocation de méthodes sur des services distants. Basé sur XML, SOAP a pour principal objectif d'assurer la communication entre machines. Le protocole permet d'appeler une méthode RPC et d'envoyer des messages aux machines distantes via HTTP. Ce protocole est très bien adapté à l'utilisation des services Web, car il permet de fournir au client une grande quantité d'informations récupérées sur un réseau de serveurs tiers, voyez :



SOAP est bien plus populaire et utilisé que XML-RPC. C'est une recommandation du W3C. D'après cette recommandation, SOAP est destiné à être un protocole léger dont le but est d'échanger des informations structurées dans un environnement décentralisé et distribué. Une des volontés du W3C vis-à-vis de SOAP est de ne pas réinventer une nouvelle technologie. SOAP a été construit pour pouvoir être aisément porté sur toutes les plates-formes et les technologies existantes.

Structure d'un message SOAP :


Structure d'un message SOAP
Protocole Header
Protocole de transport ex : HTTP
Définit le document XML comme un message SOAP
Optionnelle : Stockage des informations spécifiques à la transaction (éléments optionnels : actor, MustUnderstand, encoding..)
SOAP Envelope
SOAP Header
Contenant des données à transporter
SOAP Body
SOAP Fault
Gestion d'erreurs

Envelope: cette partie contient le message et ses différents sous-blocs. Il s'agit du bloc racine XML. Il peut contenir un attribut encodingStyle dont la valeur est une URL vers un fichier de typage XML qui décrira les types applicables au message SOAP.
Header: c'est un bloc optionnel qui contient des informations d'en-têtes sur le message. S'il est présent, ce bloc doit toujours se trouver avant le bloc Body à l'intérieur du bloc Envelope.
Body: est le bloc qui contient le corps du message. Il doit absolument être présent de manière unique dans chaque message et être contenu dans le bloc Envelope.
Fault: ce bloc sert à reporter des erreurs lors du traitement du message, ou lors de son transport. Il ne peut apparaître qu'une seule fois par message. Sa présence n'est pas obligatoire.

3) Conception détaillée

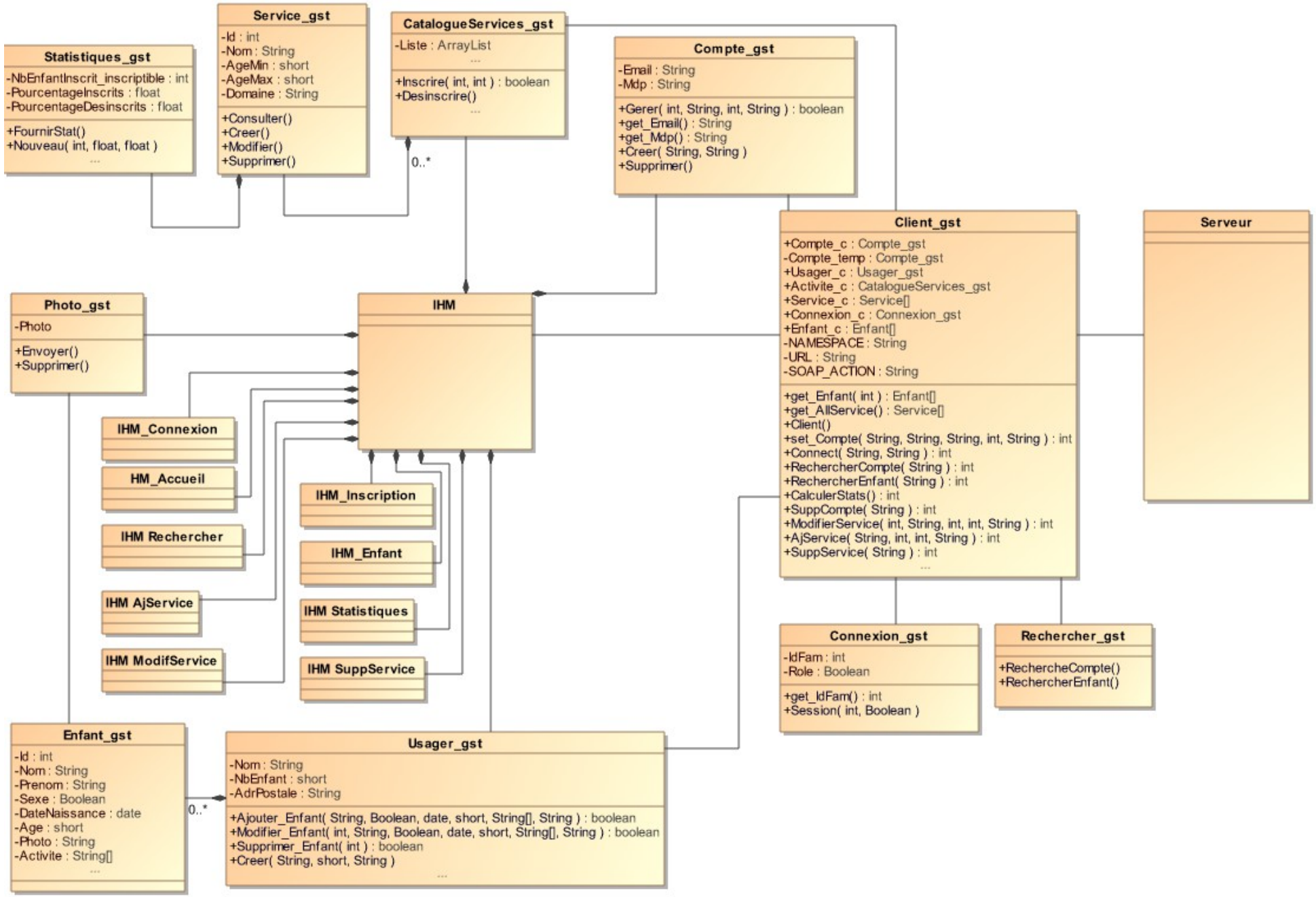
a. Diagramme de cas d'utilisation

→ *cf. Section II. 2)*

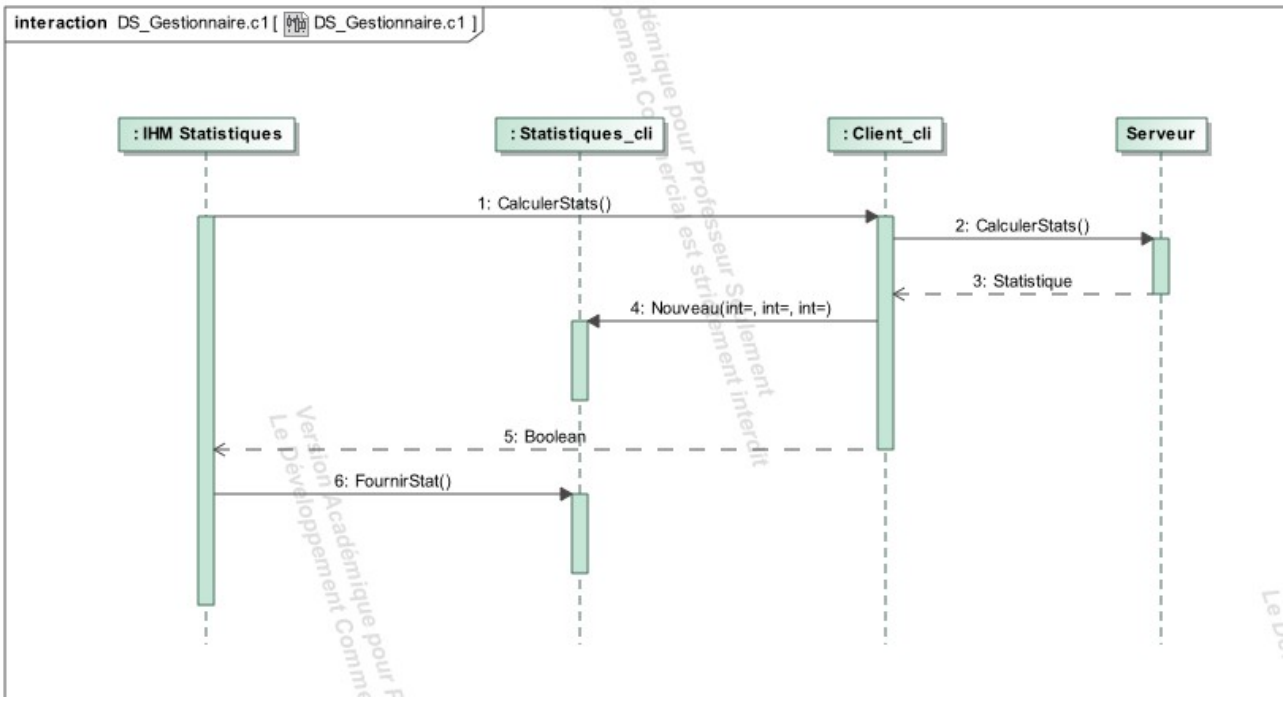
b. Diagramme de classe détaillé

Haute qualité :

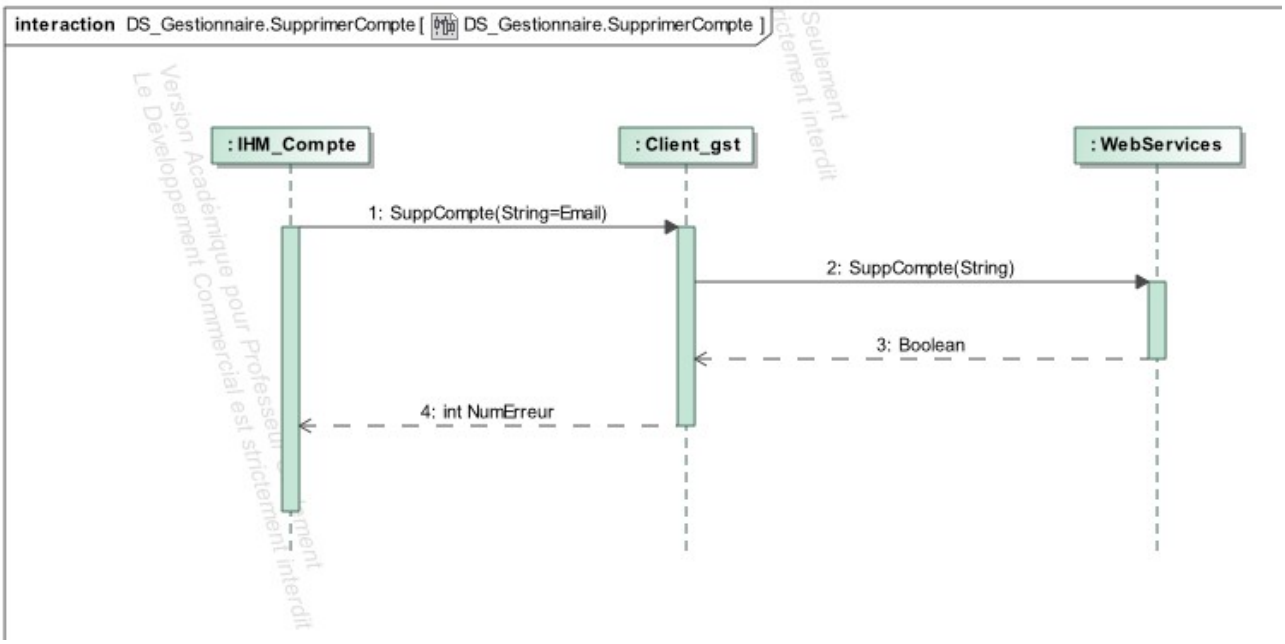
http://image.noelshack.com/fichiers/2014/22/1401538984-dc-gestionnaire.png


Service_gst
-Id : int
-Nom : String
-AgeMin : short
-AgeMax : short
-Domaine : String
+Consulter()
+Creer()
+Modifier()
+Supprimer()
CatalogueServices_gst
-Liste : ArrayList
...
+Inscrire(int, int) : boolean
+Desinscrire()
...
Statistiques_gst
-NbEnfantInscrit_inscriptible : int
-PourcentageInscrits : float
-PourcentageDesinscrits : float
+FournirStat()
+Nouveau(int, float, float)
...
Compte_gst
-Email : String
-Mdp : String
+Gerer(int, String, int, String) : boolean
+get_Email() : String
+get_Mdp() : String
+Creer(String, String)
+Supprimer()
0..*
Client_gst
+Compte_c : Compte_gst
-Compte_temp : Compte_gst
+Usager_c : Usager_gst
+Activite_c : CatalogueServices_gst
+Service_c : Service[]
+Connexion_c : Connexion_gst
+Enfant_c : Enfant[]
-NAMESPACE : String
-URL : String
-SOAP_ACTION : String
+get_Enfant(int) : Enfant[]
+get_AllService() : Service[]
+Client()
+set_Compte(String, String, String, int, String) : int
+Connect(String, String) : int
+RechercherCompte(String) : int
+RechercherEnfant(String) : int
+CalculerStats() : int
+SuppCompte(String) : int
+ModifierService(int, String, int, int, String) : int
+AjService(String, int, int, String) : int
+SuppService(String) : int
...
Serveur
Photo_gst
-Photo
+Envoyer()
+Supprimer()
IHM
IHM_Connexion
HM_Accueil
IHM_Inscription
IHM Rechercher
IHM_Enfant
IHM AjService
IHM Statistiques
IHM ModifService
IHM SuppService
Connexion_gst
-IdFam : int
-Role : Boolean
+get_IdFam() : int
+Session(int, Boolean)
Rechercher_gst
+RechercheCompte()
+RechercherEnfant()
Enfant_gst
-Id : int
-Nom : String
-Prenom : String
-Sexe : Boolean
-DateNaissance : date
-Age : short
-Photo : String
-Activite : String[]
...
0..*
Usager_gst
-Nom : String
-NbEnfant : short
-AdrPostale : String
+Ajouter_Enfant(String, Boolean, date, short, String[], String) : boolean
+Modifier_Enfant(int, String, Boolean, date, short, String[], String) : boolean
+Supprimer_Enfant(int) : boolean
+Creer(String, short, String)
...

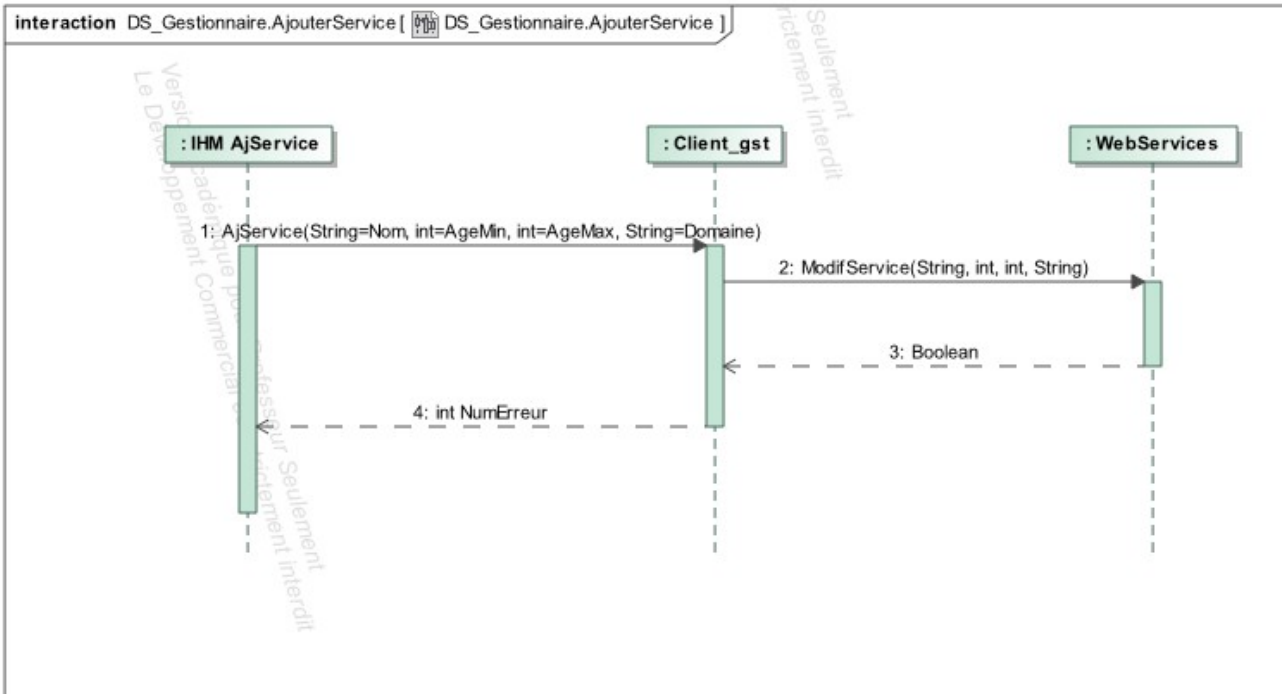
c. Diagramme séquence AfficherStat



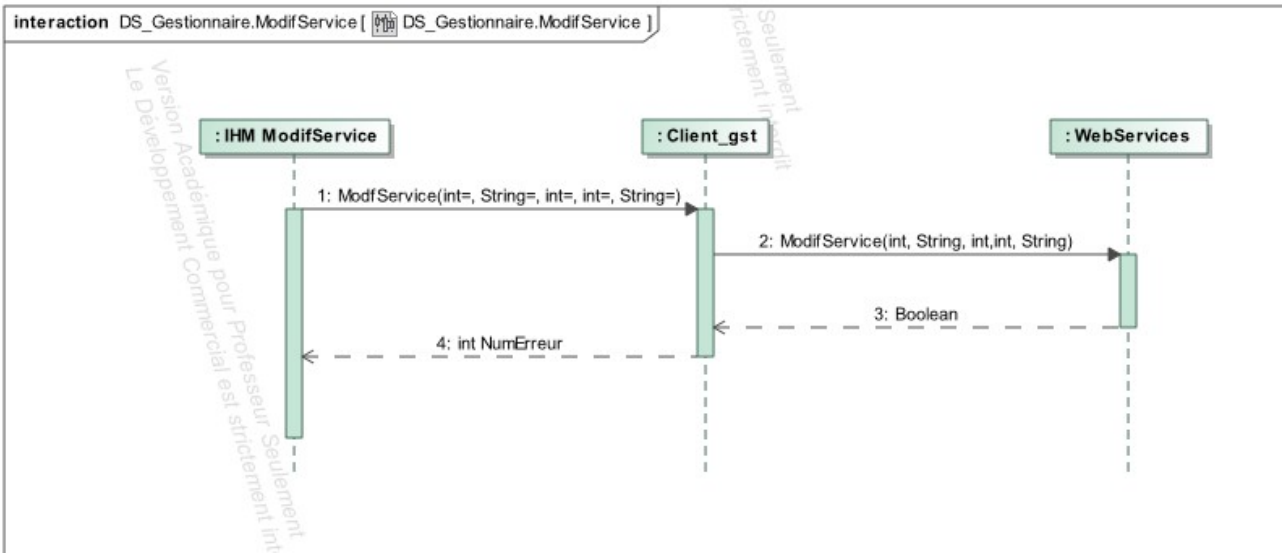
d. Diagramme de séquence SupprimerCompte



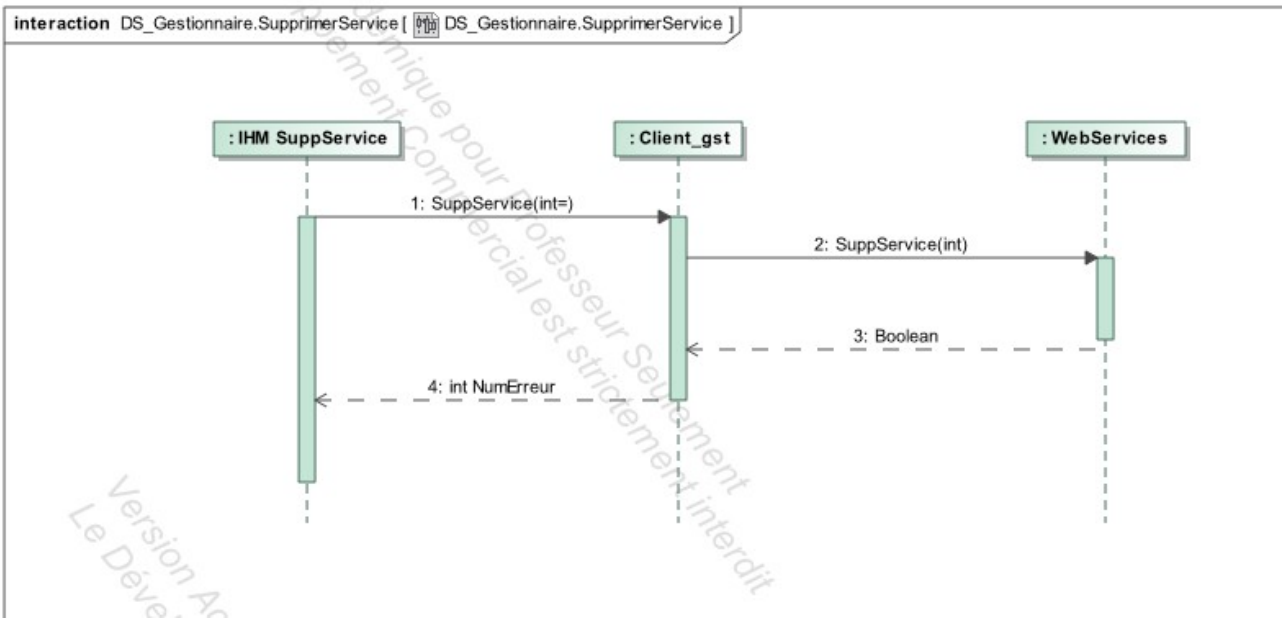
e. Diagramme de séquence AjouterService



f. Diagramme de séquence ModifService



g. Diagramme de séquence SupprimerService



Pour tous diagrammes identiques à ceux de l'usager, (DS_Connexion, DS_AjouterEnfant, etc), ce référer à la partie « Description de la partie de Romain Gaillard ».

4) Implémentation

a. Principe de fonctionnement

La classe Panneau.java hérite *(extends)* de « Application » et contient deux méthodes. C'est la structure basique dont nous avons besoin pour commencer une application JavaFX2. La méthode la plus importante est « start(Stage stage) ». Elle est automatiquement appelée lors du lancement de l'application.

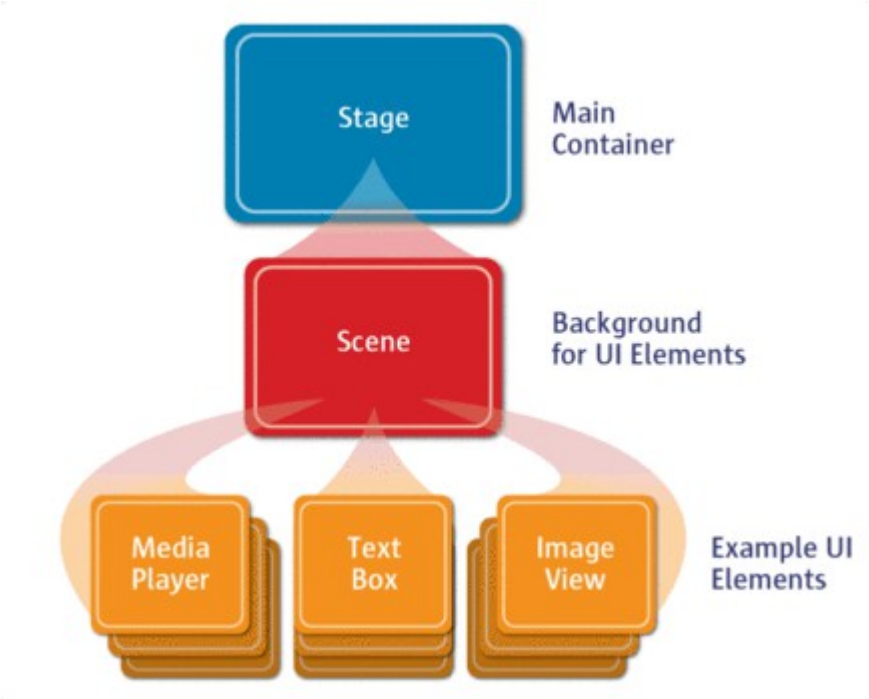


Image Source:
http://www.oracle.com/

JavaFX2 se présente comme une pièce de théâtre. Le « Stage » est le principal conteneur, habituellement représenté par une fenêtre et un menu typique (minimiser, maximiser et fermer). Dans le « Stage » se trouve les différentes « Scènes » entre lesquelles nous pouvons naviguer. A l'intérieur de chaque « Scènes » se trouve nos « Acteurs » représenté par les différents composants que nous pouvons ajouter.

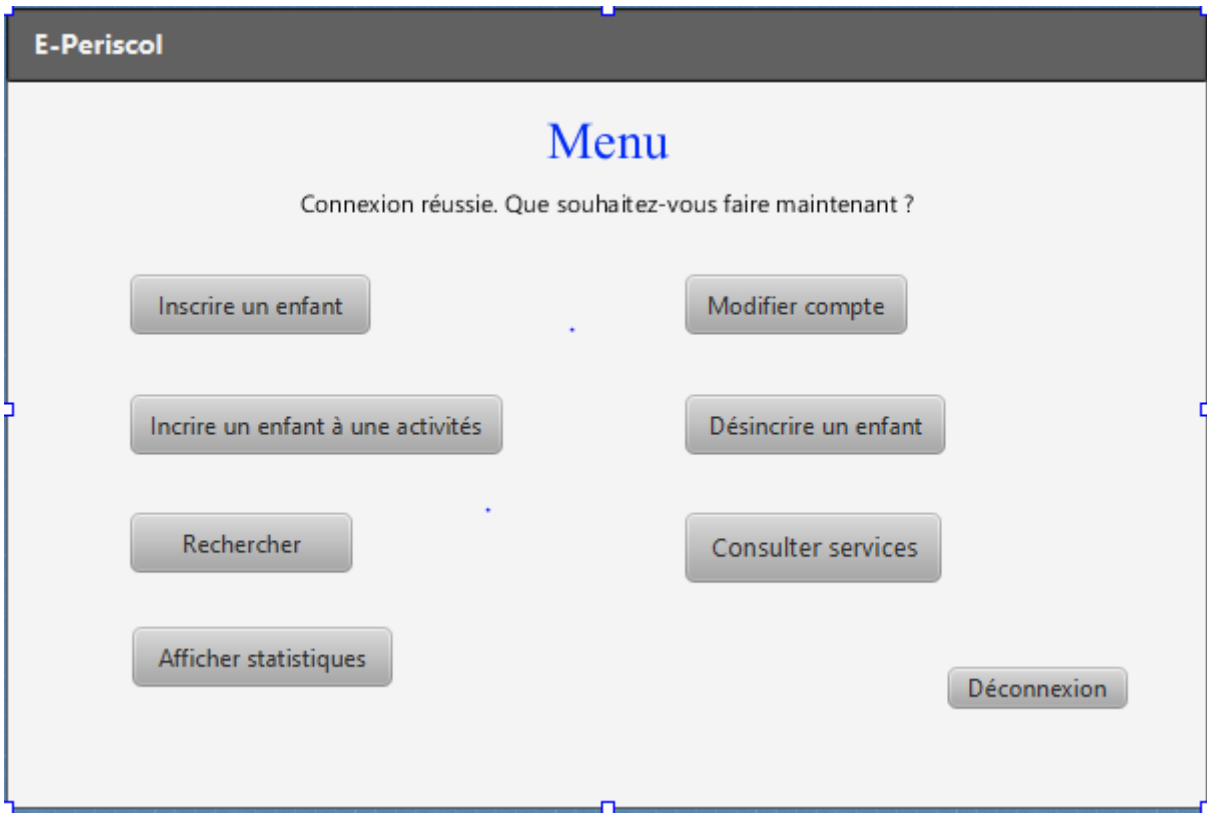
Le code suivant nous montre comment le « Stage » et la « Scène » sont chargés :
Ici le fichier Vierge.fxml est le « Stage », le fichier Accueil.fxml joue le rôle de la « Scène ».

```
public void start(Stage stage) {
             this.stage=stage;
             this.stage.setTitle("e-PeriScol");
             try {
                    //Chargement du "stage" a partir du fichier FXML
                    FXMLLoader loader = new FXMLLoader
(Main.class.getResource("Vierge.fxml"));
                    root = (BorderPane) loader.load();
                    scene =new Scene(root);
                    stage.setScene(scene);
                    stage.setHeight(400);
                    stage.setWidth(600);
                    stage.setResizable(false);
                    stage.show();
                    }catch (IOException e){e.printStackTrace(); }
             ShowAccueil();

      }
public static void ShowAccueil(){

             try {
                    //Chargement du fichier FXML contenant la scene Accueil
                    FXMLLoader loader = new FXMLLoader
(Main.class.getResource("Accueil.fxml"));
```

```
                AnchorPane overviewPage = (AnchorPane) loader.load();
                root.setCenter(overviewPage);
                page=0;
                }catch(IOException e){e.printStackTrace();}
    }
```

b. Authentification

Après s'être connecter le gestionnaire arrive sur la page principale suivante :

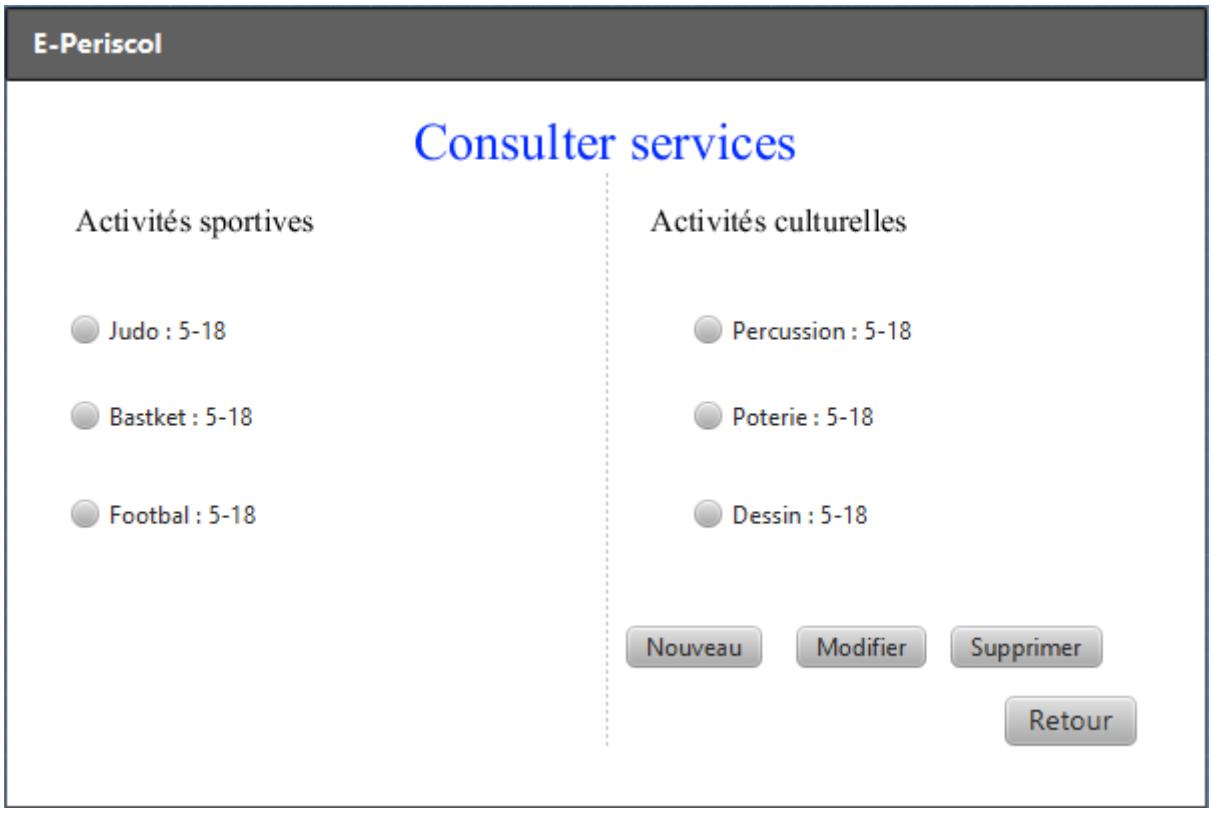


Lors du lancement de l'application, le gestionnaire doit faire face à une interface de connexion. La connexion se fait à l'aide d'un e-mail et un mot de passe unique pour le gestionnaire.

Ici débute votre inscription à l'application « E-périscol »

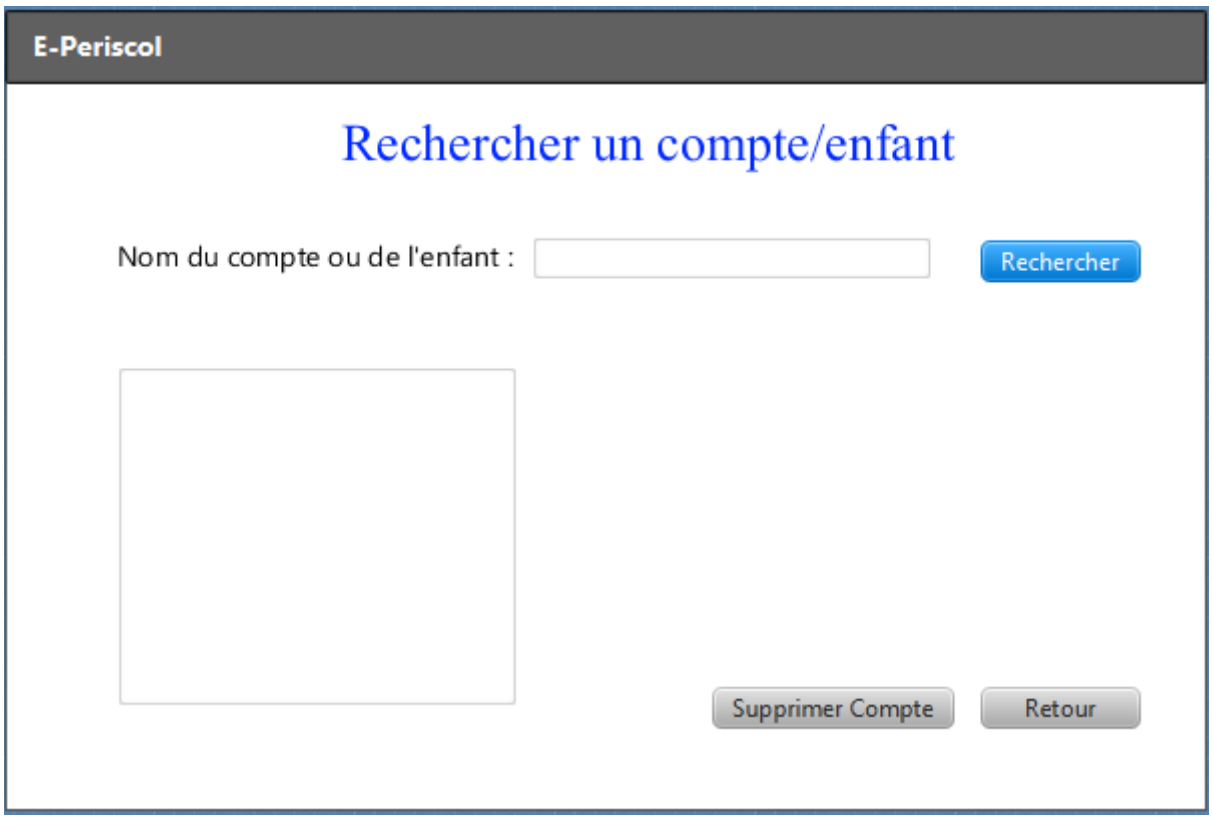
Lorsque vous avec créer un compte vous devez ajouter un premier enfant avec la fenêtre suivante :

Vous pouvez consulter un service sans même être connecté

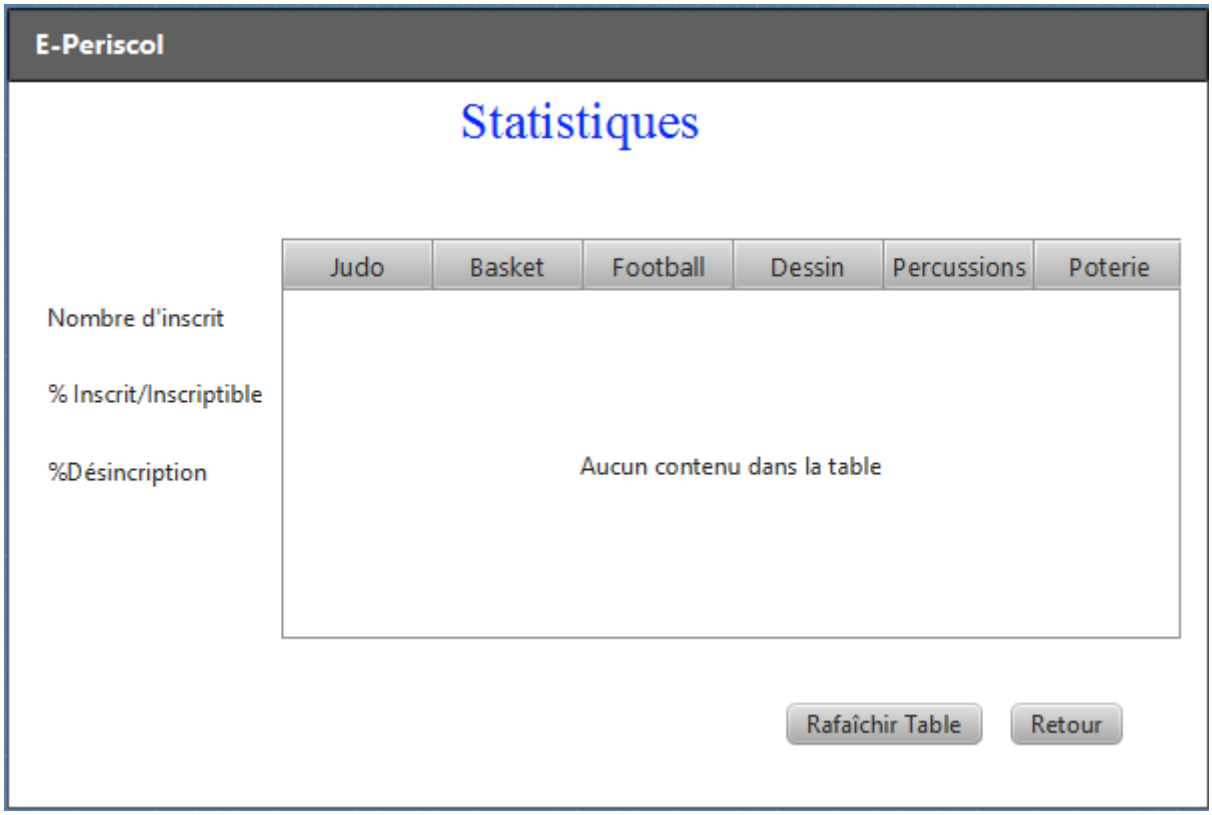


Avec la fenêtre suivante il est possible du désinscrire un enfant d'une activité périscolaire :

Le gestionnaire a des fonctionnalités supplémentaires comme la recherche d'un compte ou d'un enfant :



Ainsi que la possibilité d'afficher les statistiques



5) Test de validation

A cause de certaines incompréhensions de départ sur les modules technologiques à utiliser, j'ai pris du retard et les tests de validation n'ont pas été réalisés. Ce qu'il y a de positif est que j'ai compris comment opérer les choix technologiques et surtout comment les utiliser dans un développement.

6) Bilan personnel

Ce projet m'a permis de me familiariser avec une bibliothèque graphique pour Java très actuelle, JavaFX2, qui permet la création d'interfaces graphiques . J'ai aussi approfondi la programmation en langage Java et découvert le protocole SOAP.
Ce projet m'a aussi appris à travailler avec une équipe soudée, qui m'a épaulée et permis de passer un cap dans mon projet, même s'il n'est pas terminé.

VII. Partie personnelle – Baptiste CAHUZIERE – Base de données

1) Description de la partie personnelle

Création de la base de données avec PHPMYADMIN. La base de données doit contenir toutes les informations nécessaires à la création d'un nouveau compte usager et à l'accès aux informations pour l'administrateur (gestionnaire de mairie). C'est grâce à l'accès à cette base via le serveur que tout utilisateur pourra s'inscrire depuis n'importe quel terminal à l'activité de son choix. Pour cela le serveur et la base de données doivent être connectés, JDBC (Java DataBase Connectivity) assure cette connexion.

2) Mise en œuvre

a. Installation de la base de données

J'ai décrit le modèle d'information de la base de données à l'aide des entités-associations, adaptées à la conception des bases de données et dont est issue UML pour les diagrammes de classes :

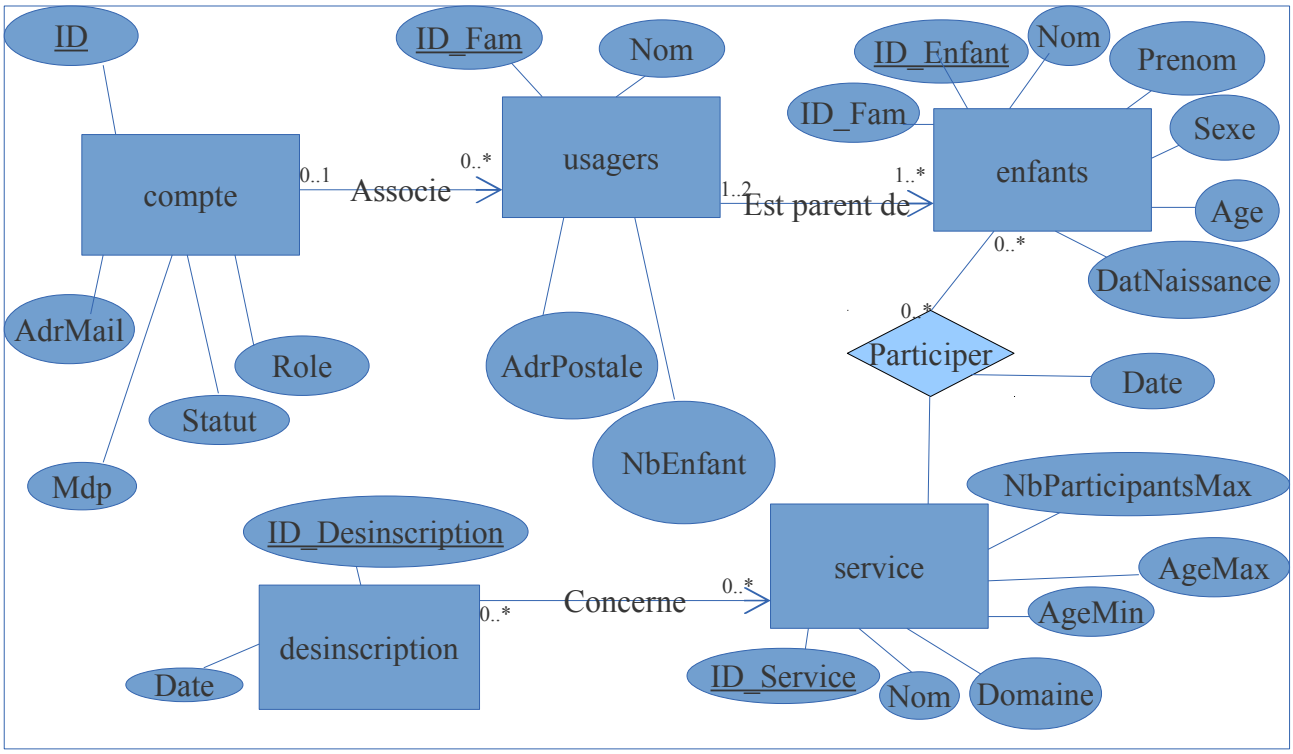


Schéma relationnel :

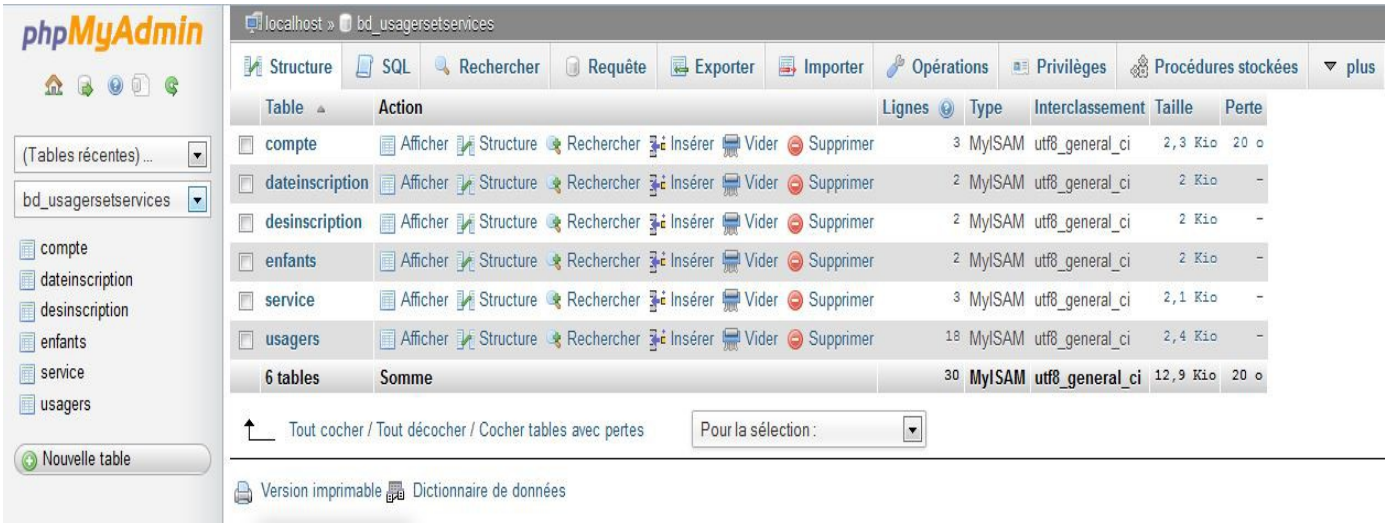
```
compte (ID, AdrMail, Mdp, Statut, Role)
usagers (ID_Fam, Nom, AdrPostale, NbEnfant)
enfants (ID_Enfant, ID_Fam, Nom, Prenom, Sexe, Age, DatNaissance)
service (ID_Service, Nom, Domaine, AgeMin, AgeMax, NbParticipantsMax)
desinscription (ID_Desinscription, Date)
Participer(ID_Enfant, ID_Service, Date)
```

La base de données du projet e-PeriScol est une base de données MySQL, Réalisée grâce au logiciel

Wampserver. En plus d'intégrer une base de données MySQL, elle intègre aussi un serveur « Apache » et une application WEB de gestion de base de données MySQL « phpMyAdmin ».

Mise en place de la base de données sur PhpMyAdmin :

Nous pouvons voir ci-dessous les différentes tables qui composent la base de données.



Sur la gauche nous avons les différentes bases déjà créées et au centre nous avons la bd_usagersetservices, celle que nous utilisons pour le projet e-PeriScol.

L'interclassement utf8_general_ci est un interclassement multilingue, insensible à la casse.

Le type MyISAM est l'ancien moteur de stockage de MySQL, il a été remplacé par InnoDB depuis la version 5.5. MyISAM dispose de l'indexation en plein texte qui permet des recherches précises et performantes sur des colonnes de type texte par des mots-clés, ainsi qu'un tri par pertinence.

Présentation des différentes tables et leurs attributs :

Table "compte" :

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	ID	int(11)			Non	*Aucune*	AUTO_INCREMENT
2	AdrMail	varchar(100)	utf8_general_ci		Non	*Aucune*	
3	Mdp	varchar(256)	utf8_general_ci		Non	*Aucune*	
4	Role	tinyint(1)			Non	*Aucune*	
5	Statut	tinyint(1)			Non	*Aucune*	

Table "dateinscription" :

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	ID_Inscription	int(11)			Non	*Aucune*	AUTO_INCREMENT	Modifier Supprimer Affiche les valeurs distinctes plus
2	Date	varchar(50)	utf8_general_ci		Non	*Aucune*		Modifier Supprimer Affiche les valeurs distinctes plus
3	ID_Enfant	int(11)			Non	*Aucune*		Modifier Supprimer Affiche les valeurs distinctes plus
4	ID_Service	int(11)			Non	*Aucune*		Modifier Supprimer Affiche les valeurs distinctes plus

Table "desinscription" :

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	ID_Desinscription	int(11)			Non	*Aucune*	AUTO_INCREMENT	Modifier Supprimer Affiche les valeurs distinctes plus
2	Date	varchar(50)	utf8_general_ci		Non	*Aucune*		Modifier Supprimer Affiche les valeurs distinctes plus

Table "enfants" :

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	ID_Enfant	int(11)			Non	*Aucune*	AUTO_INCREMENT	Modifier Supprimer Affiche les valeurs distinctes plus
2	Nom	varchar(40)	utf8_general_ci		Non	*Aucune*		Modifier Supprimer Affiche les valeurs distinctes plus
3	Prenom	varchar(40)	utf8_general_ci		Non	*Aucune*		Modifier Supprimer Affiche les valeurs distinctes plus
4	Sexe	tinyint(1)			Non	*Aucune*		Modifier Supprimer Affiche les valeurs distinctes plus
5	DateNaissance	varchar(50)	utf8_general_ci		Non	*Aucune*		Modifier Supprimer Affiche les valeurs distinctes plus
6	Photo	longblob			Non	*Aucune*		Modifier Supprimer Affiche les valeurs distinctes plus
7	ID_Fam	int(11)			Non	*Aucune*		Modifier Supprimer Affiche les valeurs distinctes plus

Table "service" :

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	ID_Service	int(11)			Non	*Aucune*	AUTO_INCREMENT
2	Nom	varchar(40)	utf8_general_ci		Non	*Aucune*	
3	Domaine	enum('Culturel', 'Sportif')	utf8_general_ci		Non	*Aucune*	
4	AgeMin	int(11)			Non	*Aucune*	
5	AgeMax	int(11)			Non	*Aucune*	
6	NbParticipantsMax	int(11)			Non	*Aucune*	

Table "usagers" :

Afficher | Structure | SQL | Rechercher | Insérer | Exporter

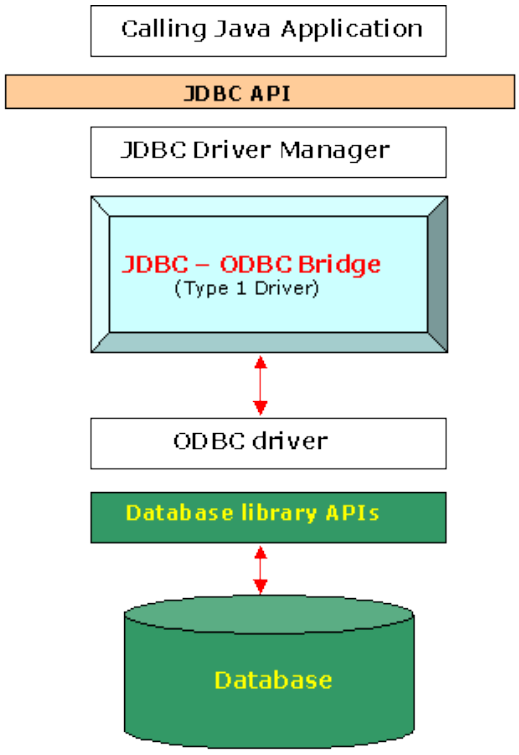
#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	ID_Fam	int(11)			Non	*Aucune*	AUTO_INCREMENT
2	AdrPostale	varchar(255)	utf8_general_ci		Non	*Aucune*	
3	NbEnfant	smallint(6)			Non	*Aucune*	
4	Nom	varchar(40)	utf8_general_ci		Non	*Aucune*	

b. Interactions avec le serveur

La base de données désormais créée est prête à accueillir les informations remplies par les clients sur leurs postes informatiques ou encore smartphones. Pour cela, il faut que le serveur Jboss et la base soient connectés. Le serveur étant un serveur d'applications java nous utiliserons JDBC pour créer cette connexion. Une fois la connexion établie, les différentes requêtes recueillies par le serveur sont interprétées et traitées . Deux types de requêtes peuvent être rencontrées, celles de consultation de la base et celles de modification de celle-ci. Par conséquent le serveur va agir de deux manières différentes suivant la demande du client. La requête une fois envoyée à la base, va donner lieu à une réponse offerte par celle-ci. La réponse est envoyée au serveur, il l'interprète et renvoie le résultat sur l'application du client. Si la demande est intraitable, un message d'erreur sera renvoyé.

c. Fonctionnement du protocole JDBC

Le protocole JDBC (Java DataBase Connectivity) est une interface de programmation pour les programmes utilisant la plateforme Java. L'api JDBC est le standard de l'industrie pour la connectivité entre les base de données indépendantes et le langage de programmation de Java. L'api JDBC permet de faire trois choses :



- Établir une connexion avec une base de données ou accédez à n'importe quel point d'émission de données tabulaire
- Envoyer les commandes SQL
- Traiter les résultats

Il existe 4 différents types de pilotes gratuits. Dans notre cas nous utilisons le premier type de pilote qui, agissant comme une passerelle, permet l'accès à une base de données à distance, via ODBC (Open Database Connectivity).

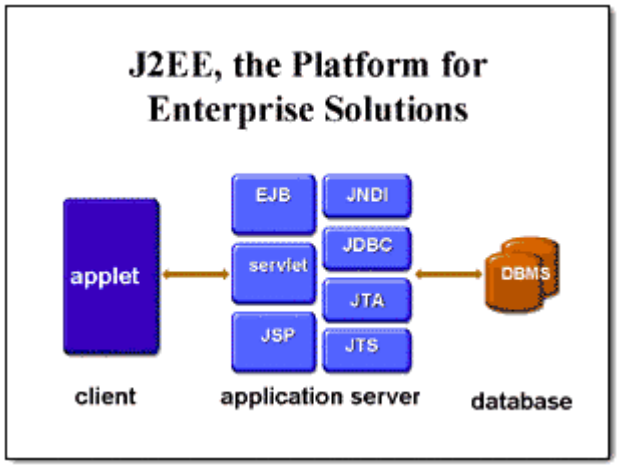
JDBC offre de nombreux avantages, notamment celui de pouvoir avoir accès à la base de données quelque soit l'endroit où elle se trouve et la technologie qu'elle utilise. JDBC cache la complexité de beaucoup de tâches d'accès aux données faisant la majeure partie du gros travail pour le programmeur. L'api JDBC est simple pour apprendre, facile à se déployer et peu coûteux à maintenir. Avec JDBC aucune configuration n'est nécessaire du côté du client.

Le driver étant écrit en Java toute l'information requise pour établir un rapport est complètement définie par l'URL de JDBC ou par un objet de point d'émission de données inscrit à un service de nomination de l'interface de Java Naming et d'annuaire (JNDI). La configuration zéro pour les clients centralise la maintenance du logiciel.

La technologie de JDBC exploite les avantages du standard Internet URLs pour identifier des connexions de base de données. L'api JDBC inclut une manière encore meilleure d'identifier et se relier à un point d'émission de données, utilisant un objet de point d'émission de données, qui

rend le code bien plus portatif et plus facile à maintenir.

Comme noyau de la plate-forme de Java 2, l'api JDBC est disponible n'importe où.
Ceci signifie que nous pouvons écrire des bases de données et y accéder n'importe où. L'api JDBC est inclus dans chacune des deux la plate-forme de Java 2, l'Édition standard (J2SE) et la plate-forme de Java 2, l'édition entreprise (J2EE), fournissant la fonctionnalité du côté serveur pour l'évolutivité industrielle de force.
Ci-contre un exemple d'un J2EE qui inclut une exécution de JDBC :



3) Conception détaillée

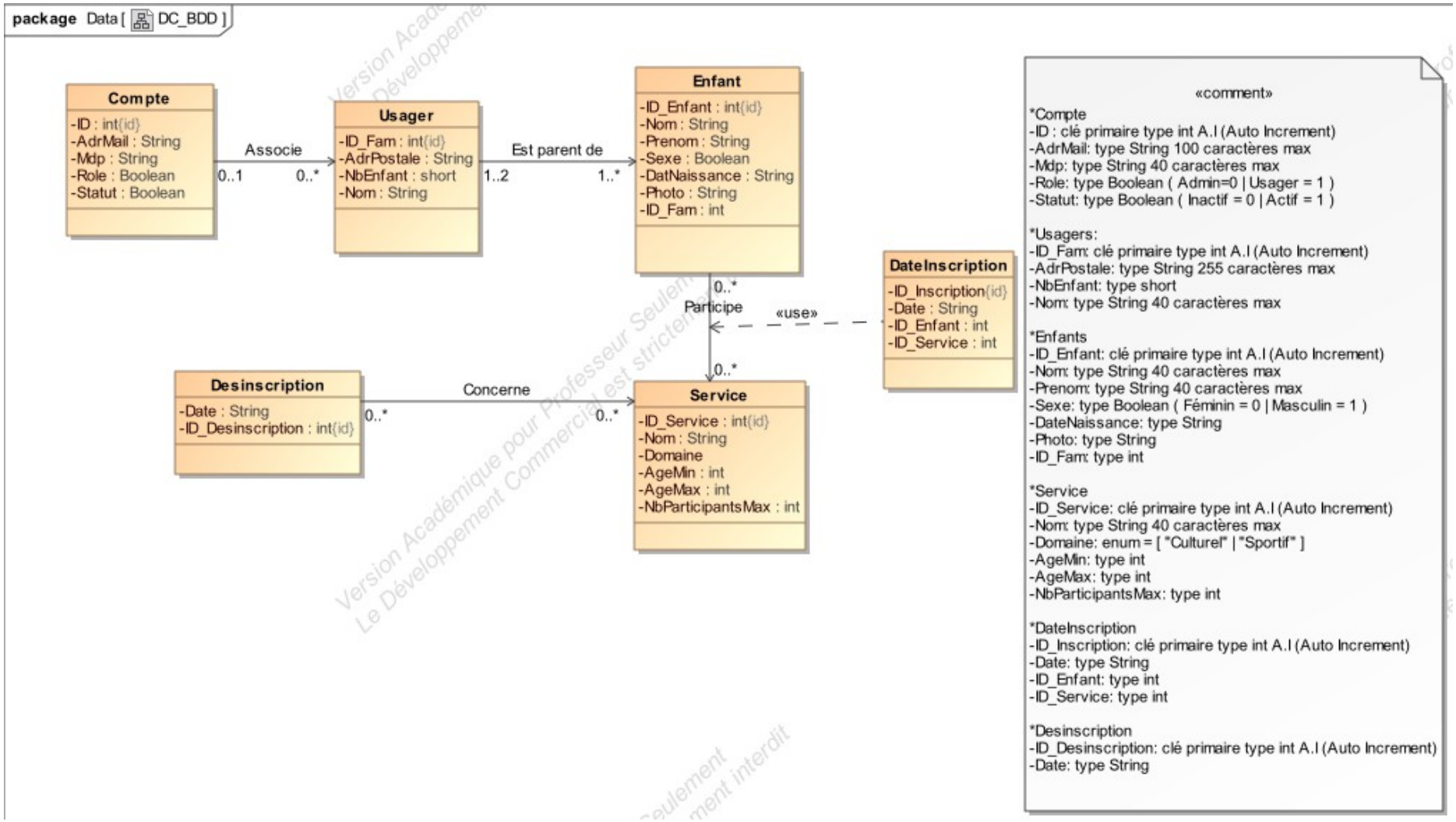
a. Modélisation UML

Avant de commencer la création de la base de données, il m'a fallu la modéliser sur le logiciel MagicDraw qui est un logiciel de modélisation UML (Unified Modeling Language). Cette action va permettre de vérifier si rien n'a été oublié d'après le cahier des charges. Dans un premier temps un diagramme de cas d'utilisation est établi, ensuite les diagrammes de classe générales et détaillées , puis viennent les diagrammes de séquences générales et détaillées.

b. Diagramme de cas d'utilisation

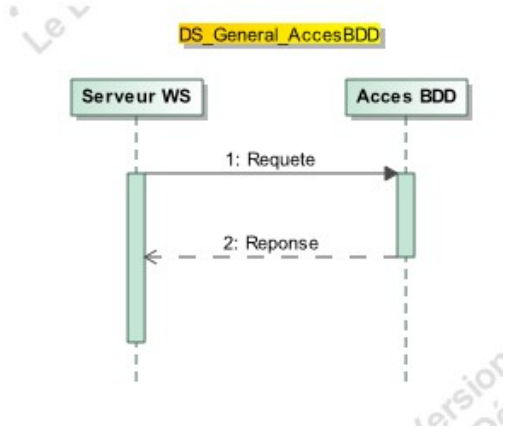
→ *cf. II. 2)*

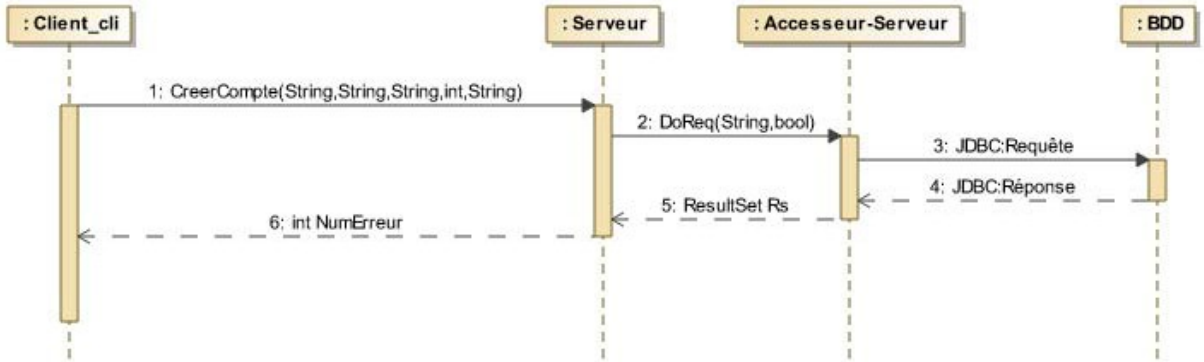
c. Diagramme de classes détaillé



d.Diagramme de séquence

Étant donné que je suis la base de données, les diagrammes de séquences sont toujours les mêmes consistant à répondre à une requête envoyée par le serveur. De ce fait je vais vous présenter un diagramme de séquence général puis un exemple avec la demande d'inscription d'un enfant à un service.





4) Implémentation

a- Connexion à la base de données

La connexion à la base de données est assurée par la fonction ci-dessous :

```
public boolean Connexion(){
          try {
               Class.forName("com.mysql.jdbc.Driver");
          } catch (ClassNotFoundException e) {e.printStackTrace();}
          try {
               co = DriverManager.getConnection(mysql,user,mdp);
               st = co.createStatement();
               return true;
          } catch (SQLException e) {System.out.println("Connection
Failed!\n");e.printStackTrace();}
          return false;
     }
```

Explications : La fonction connexion est un booléen, elle fonctionne avec JDBC. Dans un premier temps la connexion est activée grâce à la bibliothèque "java.sql.DriverManager" qui fournit le driver JDBC que nous pouvons voir dans le premier "try". Ensuite dans le second "try" nous avons deux objets, le premier est un objet de type "Connection" venant de la bibliothèque importée " java.sql.Connection" avec en paramètres "mysql, user, mdp". Mysql va être l'endroit où est stockée ma base de données, user et mdp sont les deux variables qui vont permettre de s'identifier . Le deuxième objet est de type "Statement", il est importé de la bibliothèque "java.sql.Statement". Il est fourni par l'objet Connection grâce à l'instruction co.createStatement(). L'objet Statement permet d'exécuter des instructions SQL, il interroge la base de données et retourne les résultats. Ici il exécute la connexion. Si la connexion est effectuée, le booléen retourne "true" sinon " false" avec un message d'erreur " Connection Failed !"

b- Exécution d'une requête

Il existe deux types de requêtes, celles de "consultation" de la base et celles de "modifications" de celle-ci.

```
public ResultSet Do_Req(String a, boolean b){
        if(b==false){
             try {
                     rs = st.executeQuery(a);
               if (rs != null)return rs;
           } catch (SQLException e) { e.printStackTrace(); }
        } else
             try { st.executeUpdate(a); } catch (SQLException e)
{ e.printStackTrace(); }
        return null;
```

Explications: La fonction Do_Req est de type ResultSet, cet objet est importé depuis la bibliothèque " java.sql.ResultSet ". Il permet d'effectuer diverses actions sur des résultats de

requêtes SQL. Deux paramètres sont présents, le premier est un string, c'est ici que la requête va être écrite. Le deuxième paramètre est un booléen, suivant s'il est à 1 ou 0, nous saurons si nous exécutons une requête de modification ou de consultation de la base.

Dans le premier "try" on exécute la requête grâce à executeQuery, c'est à dire qu'ici nous exécutons une requête de consultation de la base, avec en paramètre le booléen b mis à 0. Si le résultat n'est pas "null" alors la requête a bien été exécutée donc on retourne le résultat stocké dans l'objet resultSet. Sinon on fait un exécuteUpdate car cela signifie que c'est une requête de modification.

5) Test de validation

a- Connexion à la base de données

La connexion à la base de données ne fonctionnera que si les identifiants sont bons, ce qui est évident. Lorsque la connexion est réalisée, aucun message n'apparait sur le serveur, cependant si la connexion échoue à cause d'identifiants mauvais par exemple, le serveur renvoie un message d'erreur. Ici nous pouvons voir que c'est le nom d'utilisateur qui n'est pas bon.

```
.0.0.0-8080-1) Connection Failed!
.0.0.0-8080-1)

.0.0.0-8080-1) java.sql.SQLException: Access denied for user 'Testfzg'@'d306-p01.solstice2.kastler' (using password: YES)

.0.0.0-8080-1)      at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:1056)

.0.0.0-8080-1)      at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:957)

.0.0.0-8080-1)      at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3376)
```

b- Exécution d'une requête

La requête que je vais vous présenter est une requête de modification de la base, elle est donc exécutée avec un "executeUpdate". Nous allons voir comment les échanges se font entre le smartphone et le serveur et le résultat observé sur la base de données. L'objectif est d'ajouter un enfant à un service périscolaire proposé par la mairie.

```
1 0.000000000 192.168.3.3 192.168.3.5 HTTP/XML 658 POST /E-periscol/Actions_Client?WSDL HTTP/1.1
⊞ Frame 1: 658 bytes on wire (5264 bits), 658 bytes captured (5264 bits) on interface 0
⊞ Ethernet II, Src: Cisco-Li_1a:8f:d3 (c0:c1:c0:1a:8f:d3), Dst: Dell_c4:42:c6 (00:1e:4f:c4:42:c6)
⊞ Internet Protocol Version 4, Src: 192.168.3.3 (192.168.3.3), Dst: 192.168.3.5 (192.168.3.5)
⊞ Transmission Control Protocol, Src Port: 33584 (33584), Dst Port: http-alt (8080), Seq: 1, Ack: 1, Len: 592
⊞ Hypertext Transfer Protocol
⊟ eXtensible Markup Language
  ⊟ <v:Envelope
      xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:d="http://www.w3.org/2001/XMLSchema"
      xmlns:c="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:v="http://schemas.xmlsoap.org/soap/envelope/">
      <v:Header/>
    ⊟ <v:Body>
      ⊟ <n0:Get_All_Service
          id="o0"
          c:root="1"
          xmlns:n0="http://192.168.3.5:8080/Actions_Client"/>
        </v:Body>
      </v:Envelope>
```

Nous avons ici la première étape où nous pouvons voir la demande de l'utilisateur en ce qui concerne les services disponibles pour l'enfant.

Le serveur applicatif renvoie cette réponse à l'utilisateur, nous pouvons voir les services proposés dans un catalogue ainsi que les conditions à remplir pour y participer, l'âge minimum par exemple.

Voici ce que le serveur reçoit une fois que le formulaire a été rempli du côté de l'utilisateur. On peut voir que c'est la fonction AjouterEnfant qui est utilisée ici. Tous les renseignements sont visibles dans cette enveloppe. Le sexe étant à false signifie que c'est un garçon. L'enfant est aussi rattaché à l'ID de l'usager qui l'a inscrit, en l'occurrence ici c'est l'usager 104 qui a inscrit son enfant.

```
eXtensible Markup Language
 <soap:Envelope
     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <ns2:AjouterEnfantResponse
         xmlns:ns2="http://192.168.3.5:8080/Actions_Client">
       <return>
         true
         </return>
       </ns2:AjouterEnfantResponse>
     </soap:Body>
   </soap:Envelope>
```

Le serveur renvoie une réponse à l'utilisateur comme quoi l'ajout s'est bien passé.

Du côté de la base de données, elle a évolué puisqu'une ligne vient d'être ajoutée à la table Enfant :

+ Options

	ID_Enfant	Nom	Prenom	Sexe	DateNaissance	Photo	ID_Fam
Modifier Copier Effacer	1	Caup	Dorian	1	1994-12-23	[BLOB - 0 o]	1
Modifier Copier Effacer	2	c	d	0	2003-11-12	[BLOB - 0 o]	2
Modifier Copier Effacer	3	peytureau	clément	0	27/5/2014	[BLOB - 19 o]	104
Modifier Copier Effacer	4	cahuziere	baptiste	0	9/7/1993	[BLOB - 19 o]	104

6)Bilan personnel

J'ai apprécié ce projet, il était intéressant en tous points. J 'avais sans doute le poste où la programmation était la moins présente mais en travaillant avec mon coéquipier sur le serveur j'ai pu comprendre un peu plus comment fonctionnait Java. Le travail d'équipe est un mode de travail qui me convient parfaitement, le fait de partager ses idées et faire en sorte que cela fonctionne tous ensemble m'a donné envie de participer pleinement au projet. Notre application était vraiment motivante.

VIII. Partie personnelle de Romain Gaillard – Usager Android

1) Description de la partie personnelle

Création d'une application pour terminal Android *(3.2)* destinée aux parents d'élèves pour gérer l'inscription de leur enfant à des services périscolaires proposés par une mairie. Cette interface utilisateur est crée avec le SDK Android et permet les fonctions suivantes :

- Inscription aux services e-PeriScol.
- Authentification avec l'adresse courriel et le mot de passe.
- Ajout d'un enfant au compte.
- Inscription ou désinscription d'un enfants aux services sportifs et culturels.
- Afficher les services périscolaires suivis par un enfant.
- Consulter les services disponibles. *(Hors-Authentification)*
- Modification des paramètres du compte.
- Récupération du mot de passe oublié par courriel.

2) Mise en Œuvre

a. Fonctionnement de l'application

Tout d'abord l'utilisateur doit s'inscrire au web service e-Periscol, une fois inscrit crée un compte il doit entrer ses identifiants *(son adresse courriel et son mot de passe entrés lors de la création de son compte)* pour s'authentifier. L'inscription *(création du compte)* et la connexion s'effectue par une requête SOAP vers un web service situé sur un serveur Jboss *(cf. Partie Personnelle de Dorian Caup).* Ce serveur gère de la communication avec la base de donnée. Après la connexion l'utilisateur sera redirigé vers une interface d'accueil depuis laquelle il pourra, à l'aide d'un bouton, ajouter ses enfants dans l'application. Durant l'ajout d'un enfant, le formulaire propose l'inscription aux services périscolaires proposés par la mairie.

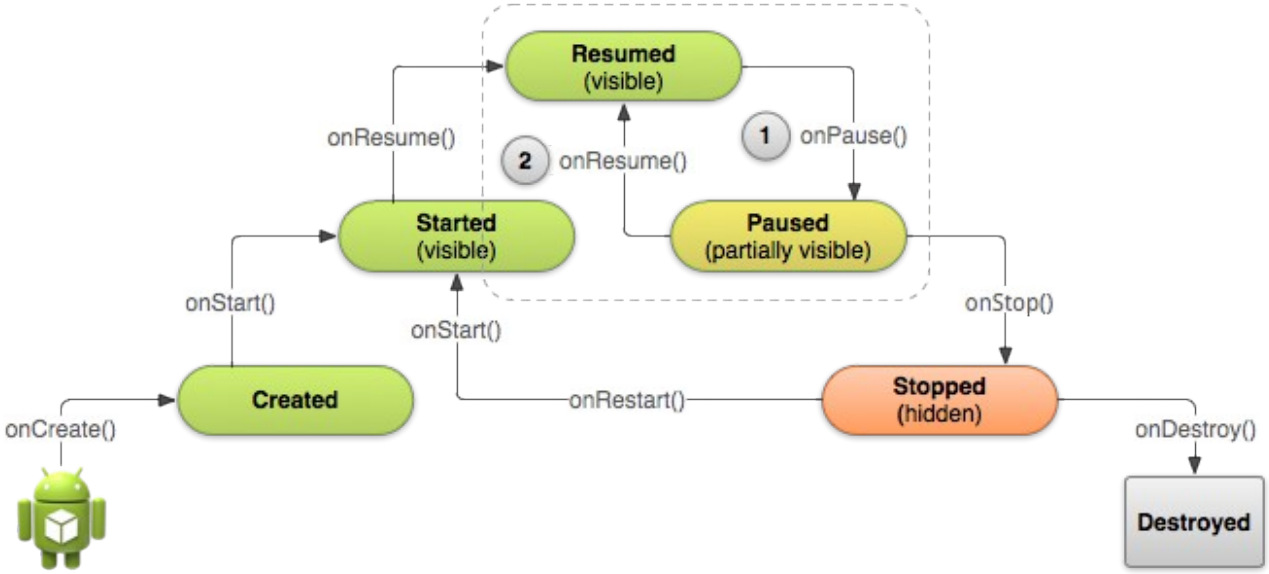
b. Environnement de développement

Android a été créé par une start-up du même nom qui a été racheté par Google en 2005. La première version d'Android a vu le jour en 2007.
Android est un système d'exploitation à part entière, basé sur le noyaux Linux. Principalement développé pour les appareils nomades et conçu pour le tactile. Extrêmement flexible, on le retrouve sur les smartphones, les tablettes, les montres, les réveils ou encore dans les voitures. Aujourd'hui 60 à 70% du parc mobile en France fonctionne sous Android.

De plus, Android est un système d'exploitation open source, offrant ainsi un fort potentiel de développement. Et par conséquent une grande diversité d'application.

Son usage tourné vers le tactile, a permis de voir la création de nouveau événement, offrant ainsi de nouvelles possibilités. Comme par exemple, faire glisser le doigt sur un écran, ou encore pivoter l'écran. Cependant, une contrainte s'impose à ce mode de fonctionnement : Le multi fenêtrage n'est pas possible. C'est pourquoi Android est conçu sur un ingénieux système d'état.

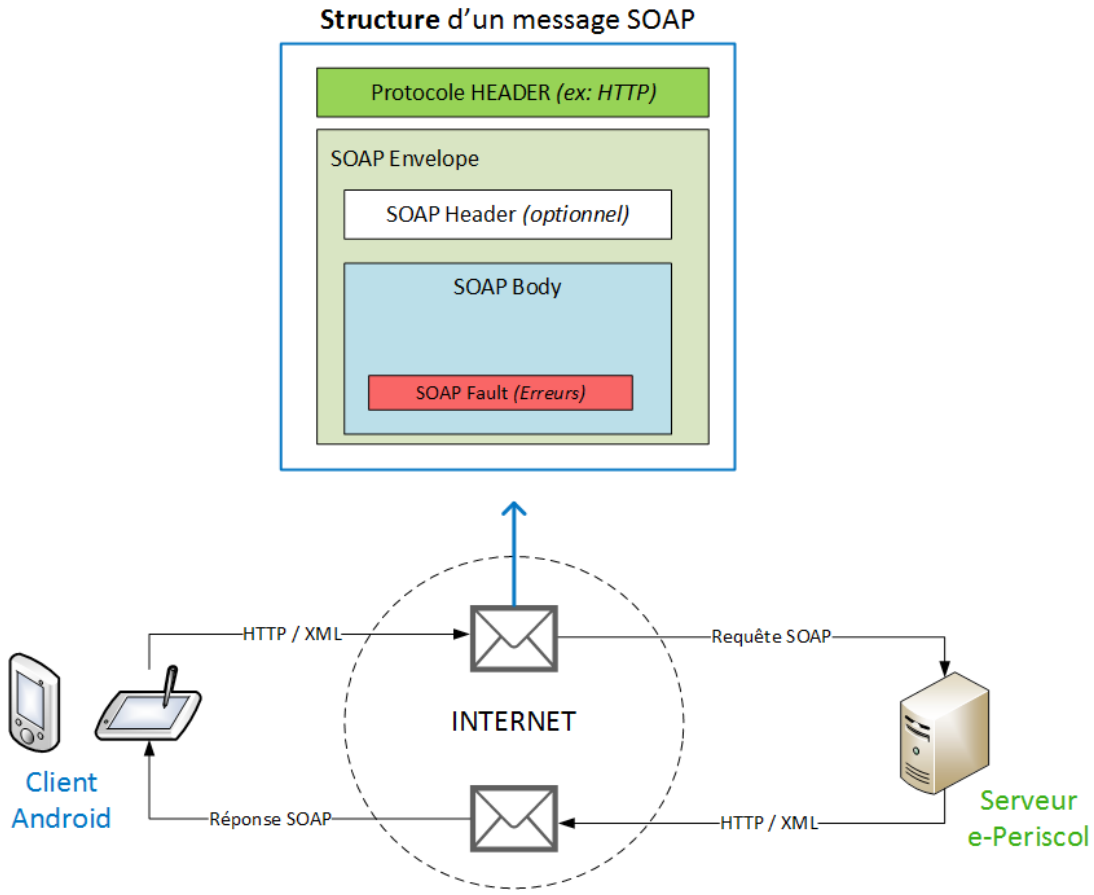
Diagramme d'état d'une application Android :



Lorsqu'une activité est lancée, on utilise la méthode **onCreate**().
Ensuite, pour lancer une vue (IHM) on utilise la méthode **onStart**()qui permet de passer l'application au premier plan. Après l'état Started, la méthode **onResume**() est directement appelée afin de permettre à l'application de reçevoir les interactions avec l'usager.
Une fois à l'état Resumed, **onPause**() permet de remettre l'application au second plan. Cette méthode intervient lorsqu'une autre IHM reprend le premier plan. *(exemple: lors d'un appel)*
Notre interface reste active mais ne reçoit plus les interactions de l'utilisateur, elle est cachée derrière la vue principale. Cependant, notre application peut continuer à exécuter des calculs qui ne consomme pas beaucoup de CPU.
La méthode **onStop**() permet de passer l'état de l'application à Stopped, celle-ci est arrêtée et n'effectue plus de calculs. Cependant on peut relancer l'application au premier plan avec **onRestart**(). Pour finir, l'application peut également être détruite avec la méthode **onDestroy**(). Cette méthode peut être appelée par le système, contre le gré de l'application, pour libérer de l'espace mémoire par exemple.

c. Protocole de communication SOAP

c.1 Schéma de fonctionnement :



Pour faire communiquer mon application avec le serveur JBOSS j'utilise le protocole SOAP, qui permet de réaliser un service web. Un service web est un programme qui fournit des fonctionnalités afin de créer une communication ou un échange entre des terminaux et des serveurs. Voici un schéma structurel du protocole SOAP : Enveloppe

c.2 Détail du fonctionnement :

Le protocole **SOAP** (**S**imple **O**bject **A**cess **P**rotocol) est courrament utilisé dans le modèle Client-Serveur et il est bâti sur le **XML*** (**E**xtensible **M**arkup **L**anguage).
Il permet à une machine d'envoyer un message XML à une autre machine. En général, c'est un client qui émet une requête et qui reçoit une réponse de la part d'un serveur. Par exemple, lorsque l'utilisateur de l'application va essayer de se connecter, il va émettre une requête vers le serveur. Il y sera spécifié son email et son mot de passe dans un fichier XML, puis il recevra une réponse (dans un fichier XML) de la part du serveur.

SOAP est basé sur le protocole **RPC** (**R**emote **P**rocédure **C**all), il s'agit d'un protocole réseau qui permet de faire un appel de procédure depuis un poste distant vers un serveur

d'application.

Les **avantages** de **SOAP** sont qu'il est indépendant vis à vis de la plate-forme ou du langage utilisé.*(ex : Java Mobile, Objective C, etc.)* De plus il est flexible car il s'adapte aux différents protocoles de transport. *(ex : HTTP/SMTP)*

En revanche, il possède quelques **inconvénients** comme le couplage fort qu'il nécessite entre le client et le serveur. Car c'est le serveur qui défini la structure de l'échange, cf. **WSDL** (**W**eb **S**ervices **D**escription **L**anguage) qui précise le protocole de communication en listant et détaillant les méthodes que le serveur de web service propose. → cf. partie personnelle de Dorian Caup. V. 2) et Annexes.

XML *: Il s'agit d'un langage de balise extensible, où la création de ces balises est libre. Elles sont définies par l'utilisateur. Contrairement au XHTML (Extensible HyperText Markup Langage) qui est un langage de balisage pour écrire des pages web où les balises y sont prédéfinies.*

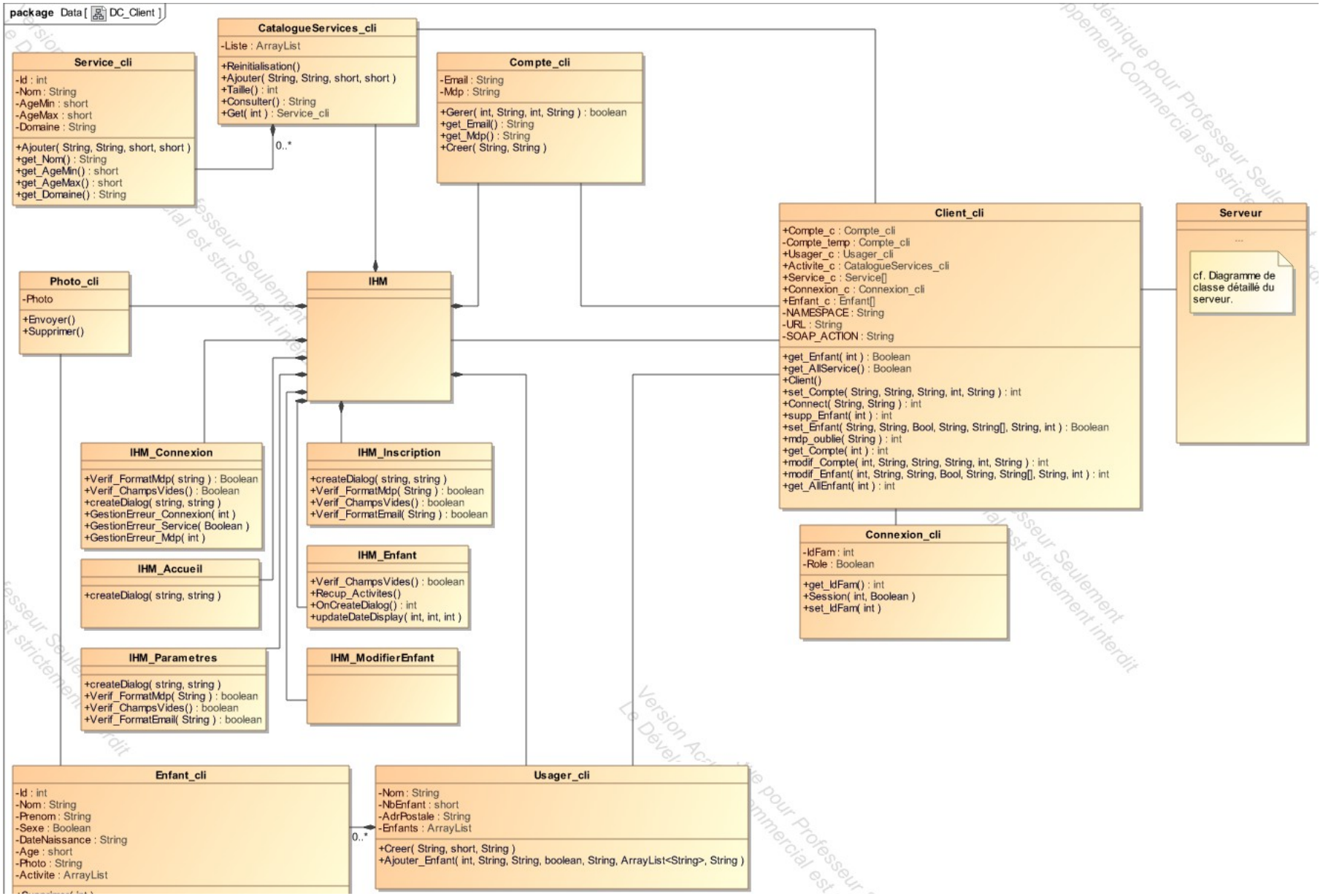
3) Conception détaillée (diagrammes)

a. Diagramme de cas d'utilisation

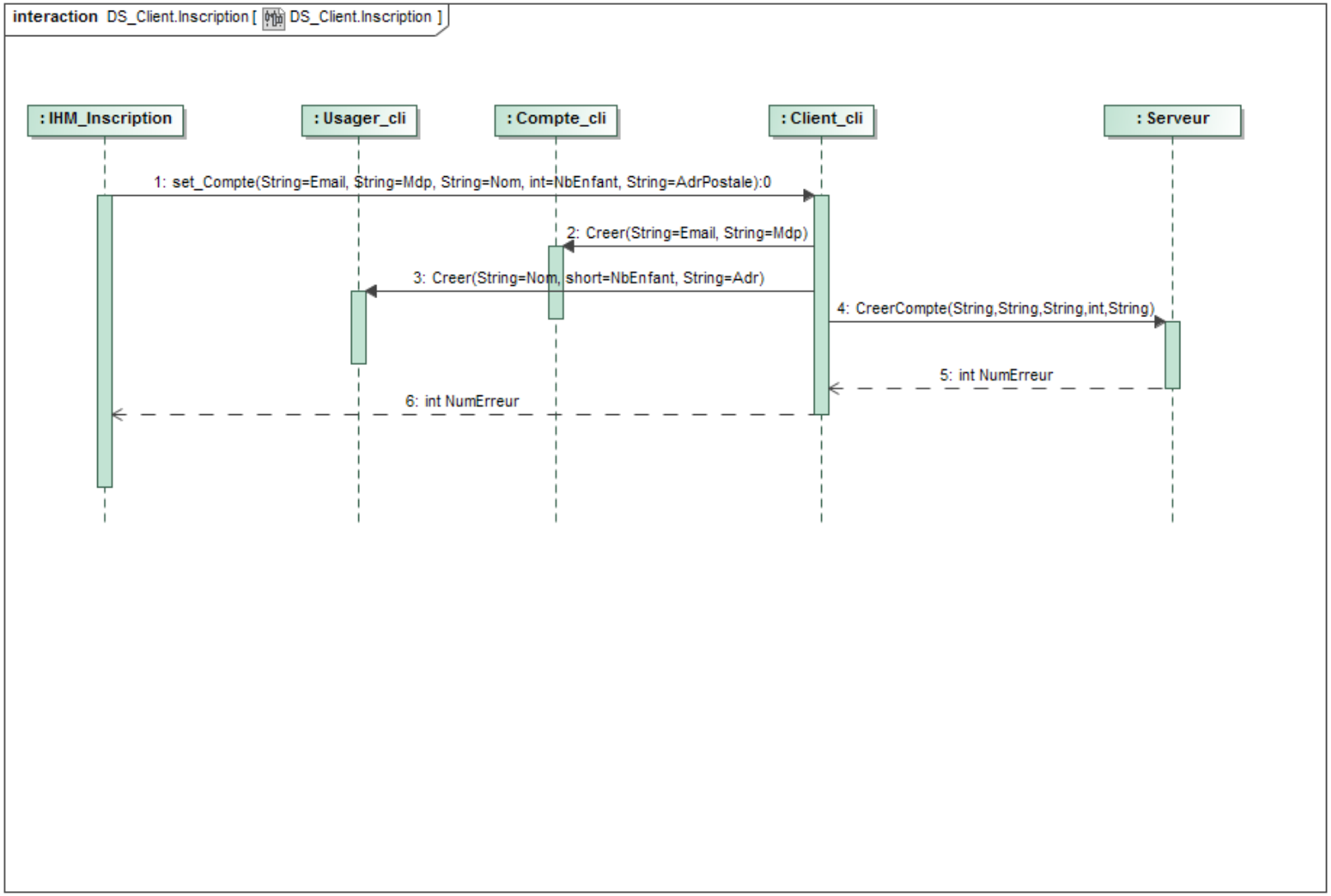
→ *cf. Section II. 2)*

b. Diagramme de classe détaillé

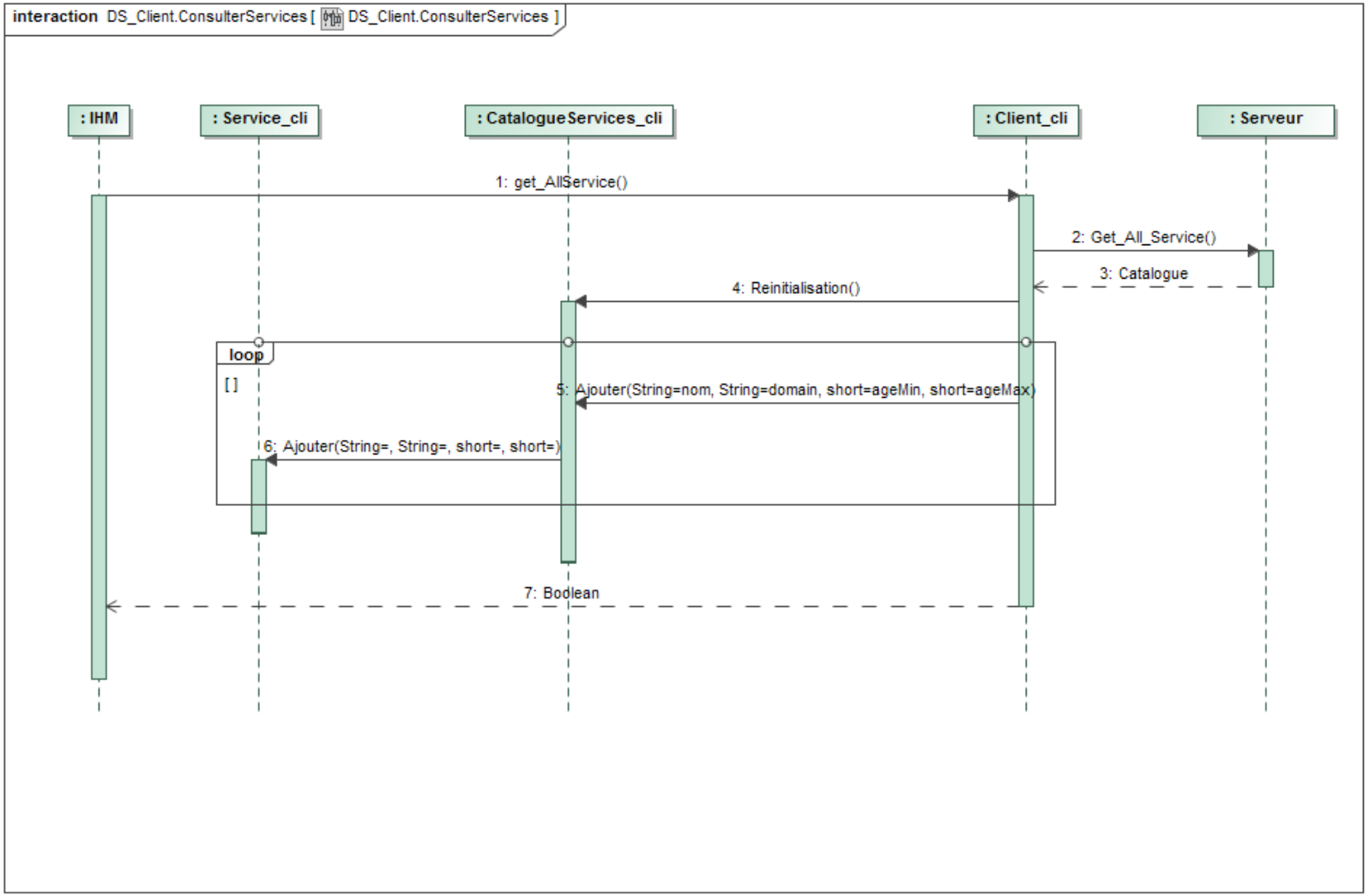
Haute qualité: http://image.noelshack.com/fichiers/2014/22/1401300913-dc-client.png


package Data [DC_Client]
CatalogueServices_cli
-Liste : ArrayList
+Reinitialisation()
+Ajouter(String, String, short, short)
+Taille() : int
+Consulter() : String
+Get(int) : Service_cli
Service_cli
-Id : int
-Nom : String
-AgeMin : short
-AgeMax : short
-Domaine : String
+Ajouter(String, String, short, short)
+get_Nom() : String
+get_AgeMin() : short
+get_AgeMax() : short
+get_Domaine() : String
0..*
Compte_cli
-Email : String
-Mdp : String
+Gerer(int, String, int, String) : boolean
+get_Email() : String
+get_Mdp() : String
+Creer(String, String)
Client_cli
+Compte_c : Compte_cli
-Compte_temp : Compte_cli
+Usager_c : Usager_cli
+Activite_c : CatalogueServices_cli
+Service_c : Service[]
+Connexion_c : Connexion_cli
+Enfant_c : Enfant[]
-NAMESPACE : String
-URL : String
-SOAP_ACTION : String
+get_Enfant(int) : Boolean
+get_AllService() : Boolean
+Client()
+set_Compte(String, String, String, int, String) : int
+Connect(String, String) : int
+supp_Enfant(int) : int
+set_Enfant(String, String, Bool, String, String[], String, int) : Boolean
+mdp_oublie(String) : int
+get_Compte(int) : int
+modif_Compte(int, String, String, String, int, String) : int
+modif_Enfant(int, String, String, Bool, String, String[], String, int) : int
+get_AllEnfant(int) : int
Serveur
...
cf. Diagramme de classe détaillé du serveur.
Photo_cli
-Photo
+Envoyer()
+Supprimer()
IHM
IHM_Connexion
+Verif_FormatMdp(string) : Boolean
+Verif_ChampsVides() : Boolean
+createDialog(string, string)
+GestionErreur_Connexion(int)
+GestionErreur_Service(Boolean)
+GestionErreur_Mdp(int)
IHM_Inscription
+createDialog(string, string)
+Verif_FormatMdp(String) : boolean
+Verif_ChampsVides() : boolean
+Verif_FormatEmail(String) : boolean
IHM_Accueil
+createDialog(string, string)
IHM_Enfant
+Verif_ChampsVides() : boolean
+Recup_Activites()
+OnCreateDialog() : int
+updateDateDisplay(int, int, int)
IHM_Parametres
+createDialog(string, string)
+Verif_FormatMdp(String) : boolean
+Verif_ChampsVides() : boolean
+Verif_FormatEmail(String) : boolean
IHM_ModifierEnfant
Connexion_cli
-IdFam : int
-Role : Boolean
+get_IdFam() : int
+Session(int, Boolean)
+set_IdFam(int)
Enfant_cli
-Id : int
-Nom : String
-Prenom : String
-Sexe : Boolean
-DateNaissance : String
-Age : short
-Photo : String
-Activite : ArrayList
0..*
Usager_cli
-Nom : String
-NbEnfant : short
-AdrPostale : String
-Enfants : ArrayList
+Creer(String, short, String)
+Ajouter_Enfant(int, String, String, boolean, String, ArrayList<String>, String)

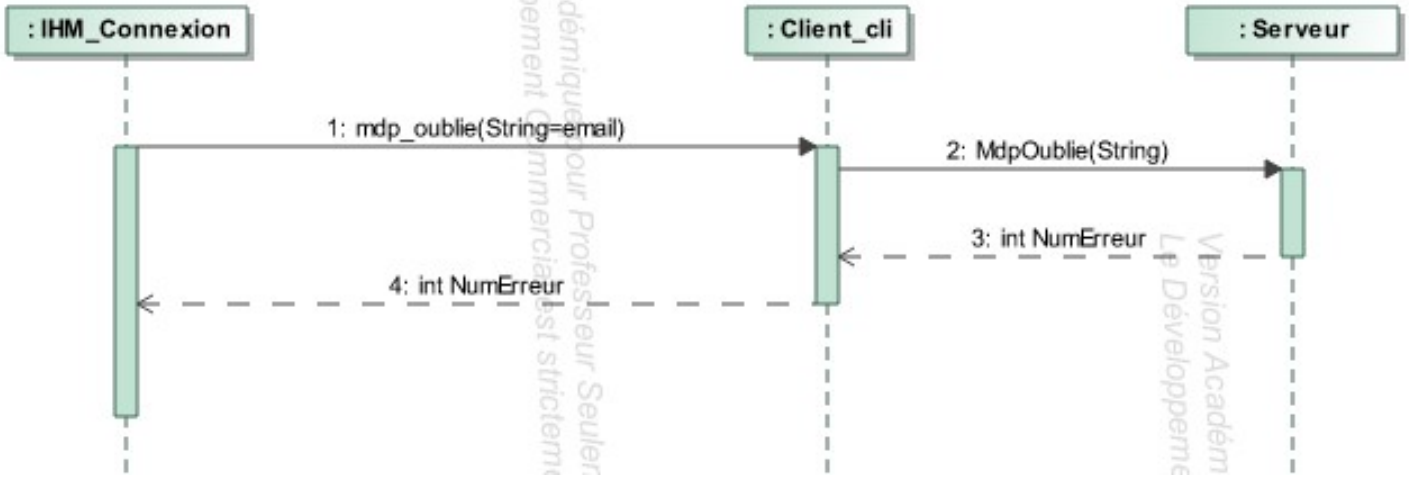
c. Diagramme de séquence (Inscription)



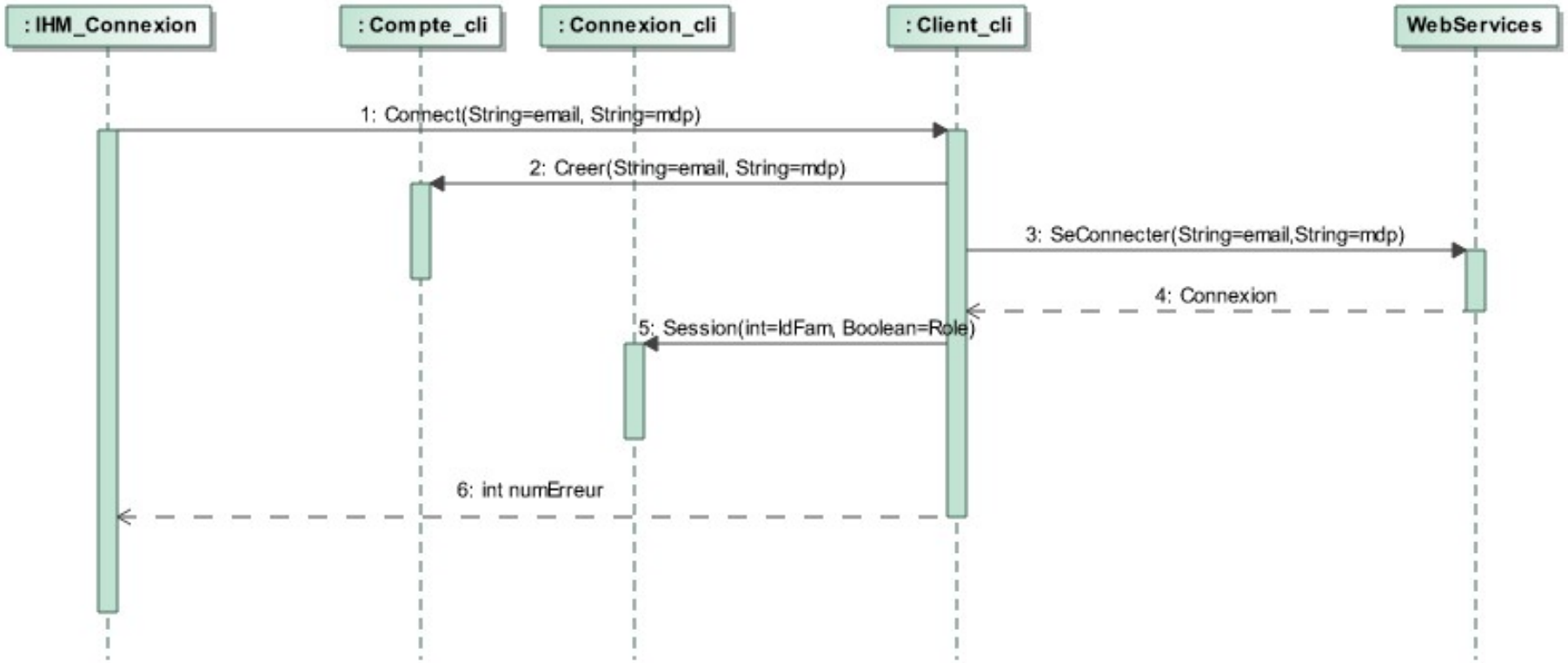
d. Diagramme de séquence (Consulter les Services)



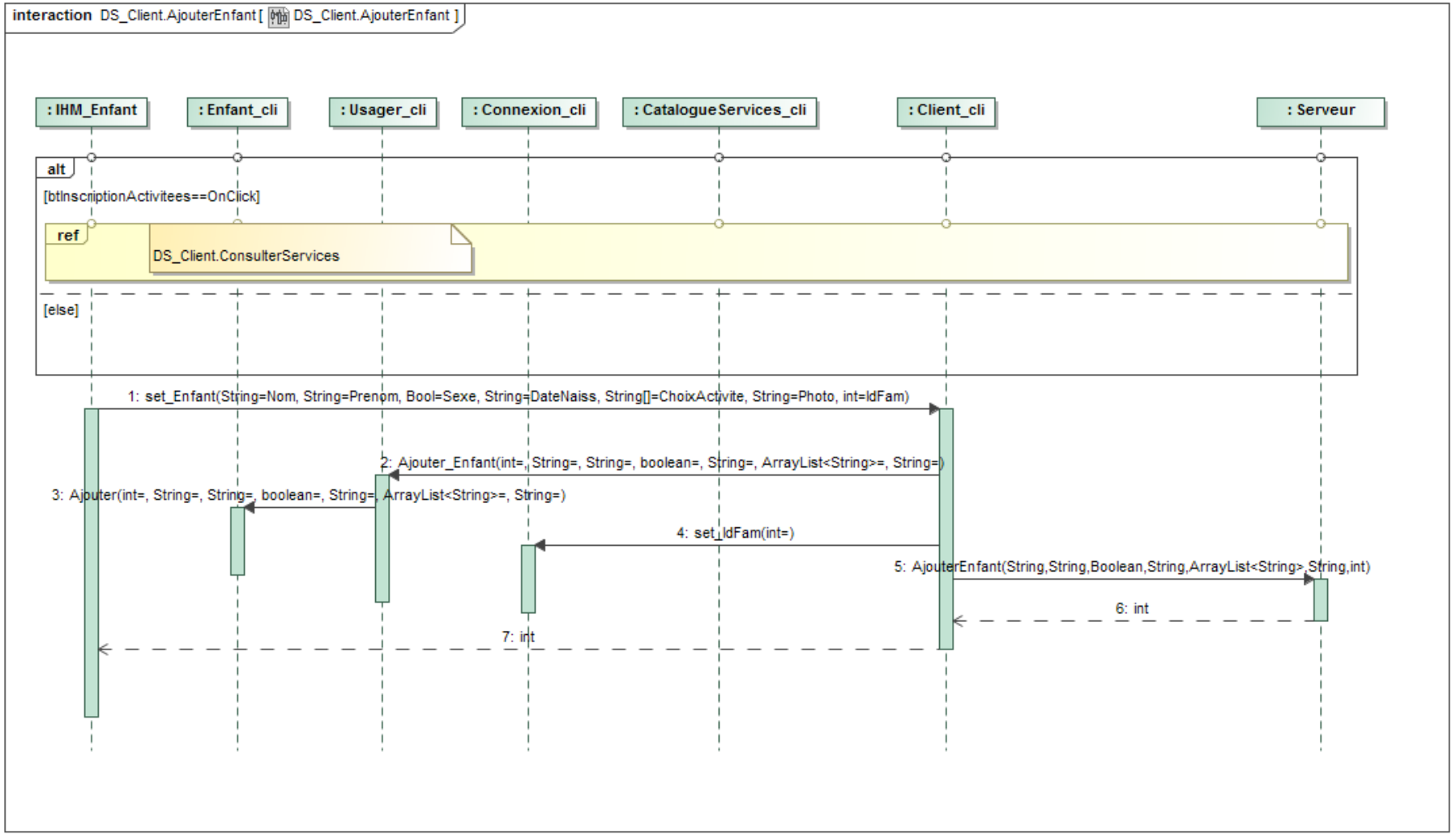
e. Diagramme de séquence (Mot de passe oublié)



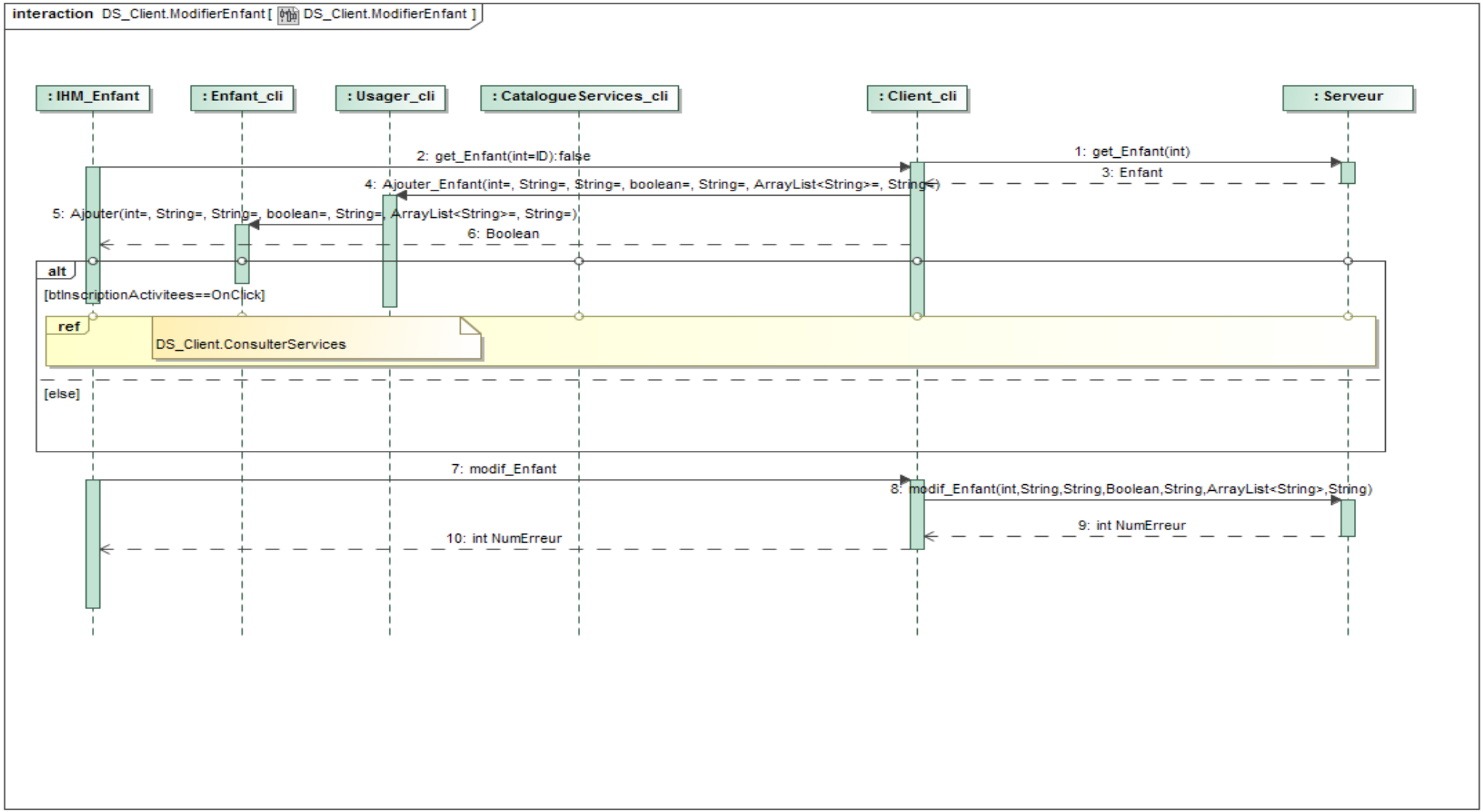
f. Diagramme de séquence (Connexion)



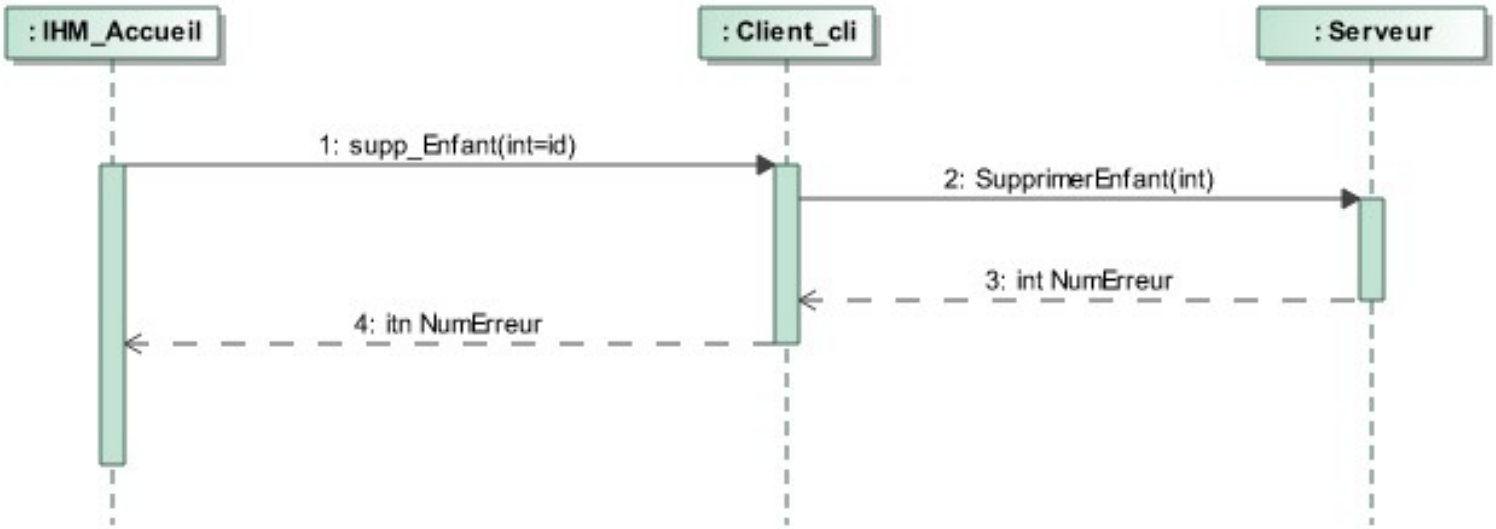
h. Diagramme de séquence (Ajouter Enfant)



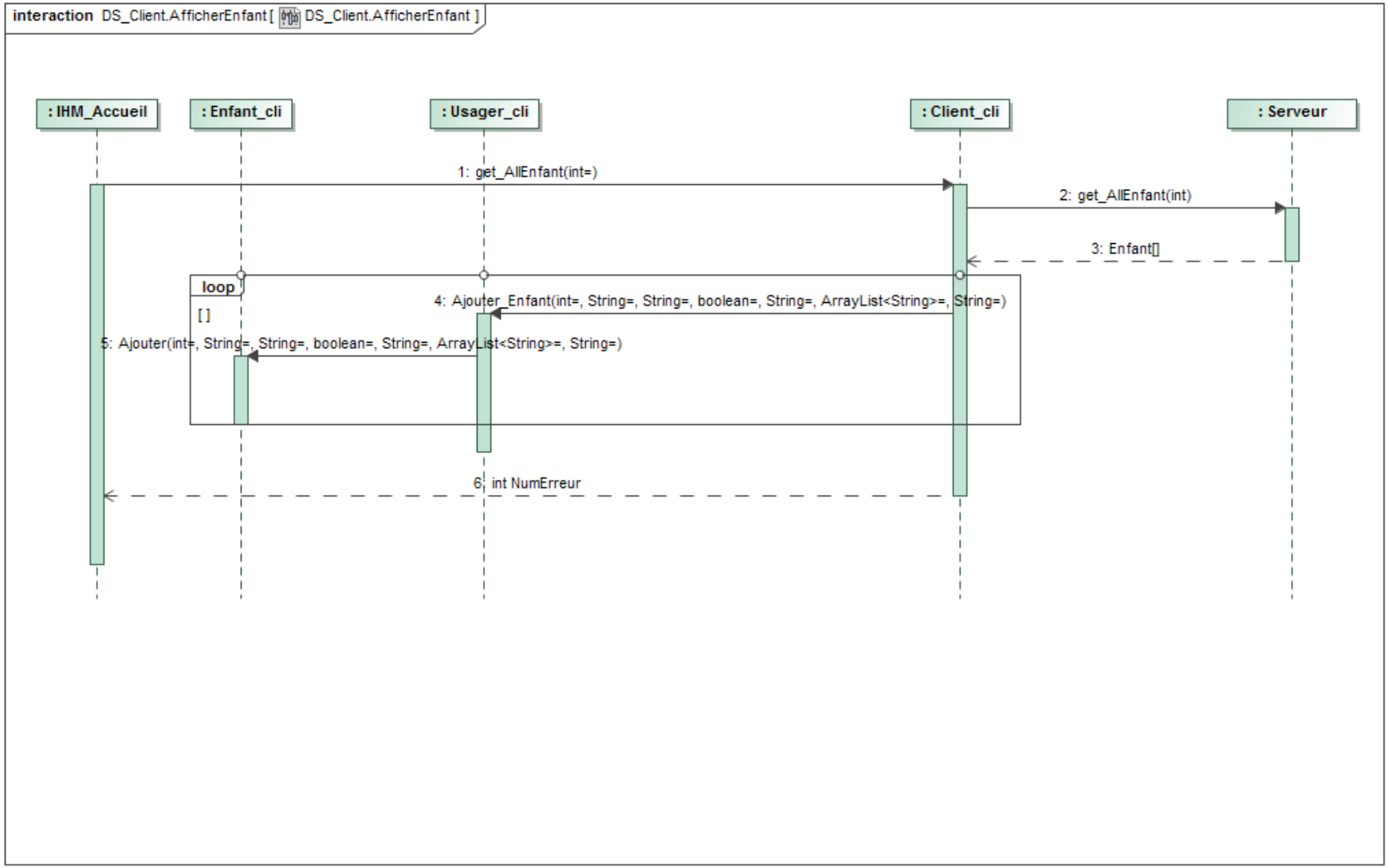
i. Diagramme de séquence (Modifier Enfant)



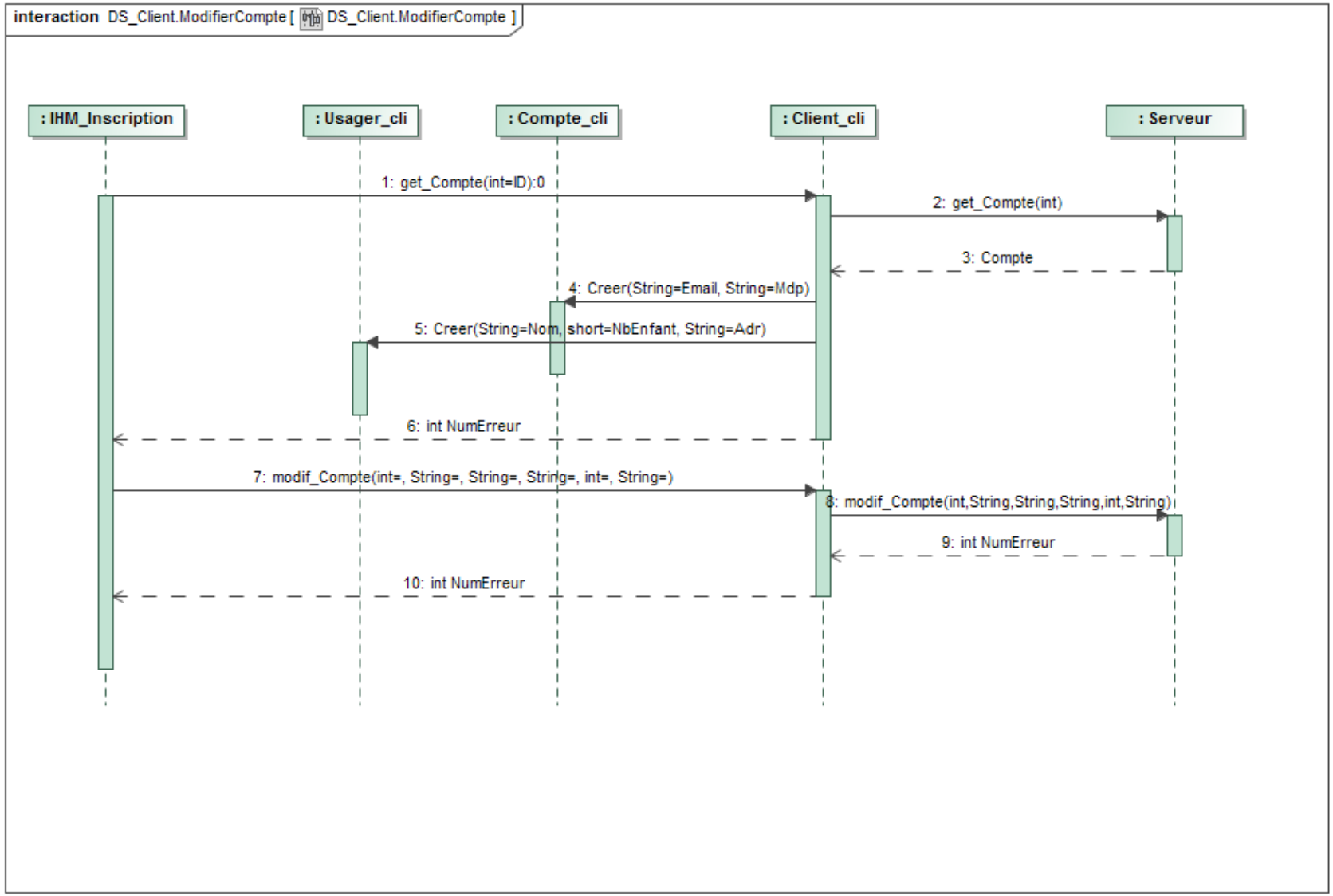
j. Diagramme de séquence (Supprimer Enfant)



k.Diagramme de séquence (Afficher les Enfants)



I. Diagramme de séquence (Modifier Compte)



4) Implémentation

Présentation des IHM (Interfaces Homme Machines)

a. IHM Connexion

Lors du lancement de l'application, la première interface permet à l'utilisateur de s'authentifier via son adresse email et son mot de passe.
Cette IHM permet également un accès à l'interface d'Inscription, via le bouton « Inscription », ainsi que la consultation des services périscolaires disponibles, et aussi la récupération du mot de passe.
→ cf. section VIII. 4) b.IHM Inscription. → cf. section VIII 4) a.4 Consulter les services
→ cf. section VIII 4) a.5 Récupération de mot de passe.

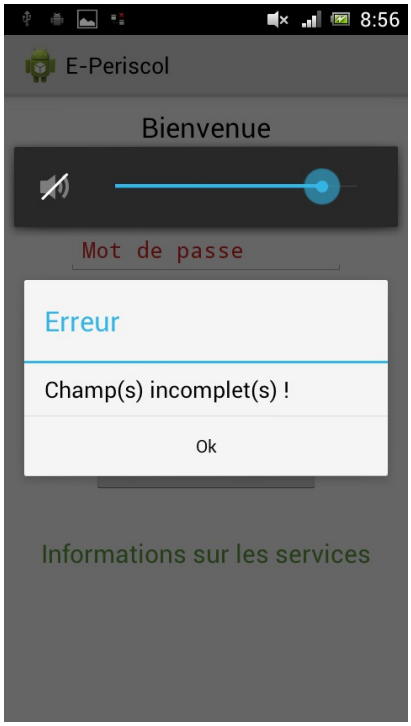
Lors de l'appui sur le bouton « Connexion », l'application fait appel à la méthode « Connect » de classe Client_cli, cet appel s'effectue dans un Thread secondaire afin de ne pas bloquer le thread principal qui affiche le chargement en cours.

```
private OnClickListener button1Listener = new OnClickListener(){
        @Override
        public void onClick(View v) {
               switch(v.getId()) {
                 case R.id.btConnexion:
                    if(Verif_ChampsVides())
                    {
                                myProgressDialog =
ProgressDialog.show(IHM_Connexion.this,"Chargement", "Veuillez patienter", true);
                                    new Thread(new Runnable() {
                                           public void run() {

                                                  NumErreur=-1;
NumErreur=client.Connect(etEmail.getText().toString(),etMdp.getText().toString());

                                myProgressDialog.dismiss();
                                Message msg = mHandler.obtainMessage(MSG_CONNEXION);
```

```
private OnClickListener button1Listener = new OnClickListener(){
         @Override
         public void onClick(View v) {
                switch(v.getId()) {
                  case R.id.btConnexion:
                     if(Verif_ChampsVides())
                     {
                                myProgressDialog =
ProgressDialog.show(IHM_Connexion.this,"Chargement", "Veuillez patienter", true);
                                    new Thread(new Runnable() {
                                           public void run() {

                                                  NumErreur=-1;
NumErreur=client.Connect(etEmail.getText().toString(),etMdp.getText().toString());

                                myProgressDialog.dismiss();
                                Message msg = mHandler.obtainMessage(MSG_CONNEXION);
                                             mHandler.sendMessage(msg);
                                           }
                                    }).start();
                       }break;
```

Explication du code :

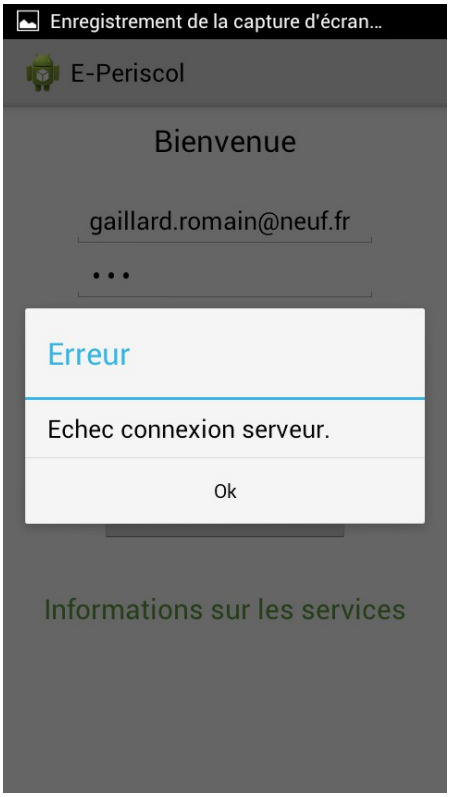
Tout d'abord, on effectue une vérification pour s'assurer que les champs sont remplis, si ce n'est pas le cas une popup d'information s'affiche.

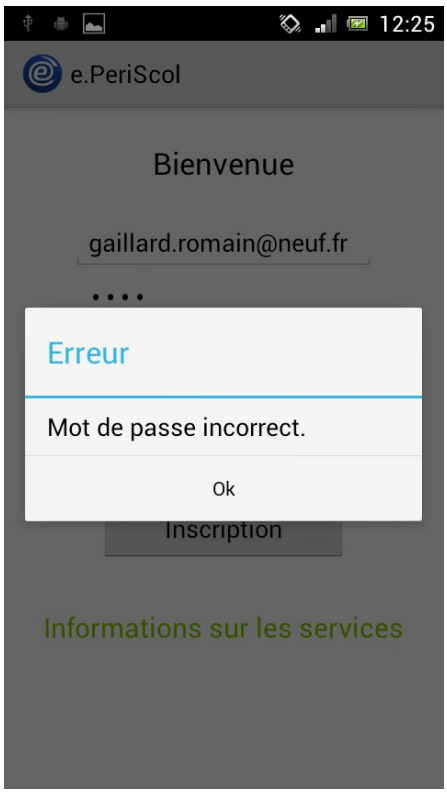


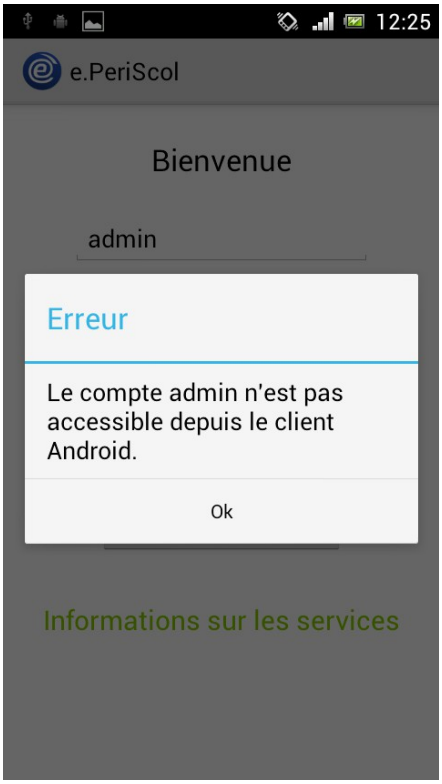
Remarque : Les champs qui sont incomplets s'affichent en rouge.

Si tous les champs sont remplis on lance une popup de chargement, avec la classe **ProgressDialog**. Ensuite, on créer le thread secondaire pour faire appel à la méthode Connect où on lui passe l'email et le mot de passe. Dès que ce thread se termine, on met fin à la popup de chargement et on traite le résultat obtenu. *(Un numéro d'erreur).* *→ Cf. Annexe du code méthode handleMessage(Message msg) et GestionErreur_Connexion(**int** numerreur)*

Voici les messages des résultats possibles :







a.1 Classe Client_cli :

Voici le code exécuté par le client lors de l'appel de la méthode Connect :

```
public int Connect(String email,String mdp){
       init_liaison();
       Compte_c.Creer(email, mdp);
       new Thread(new Runnable() {
           public void run() {
              try {
                           SoapObject request = new SoapObject(NAMESPACE,"SeConnecter");
                           request.addProperty("Email",Compte_c.get_Email());
                           request.addProperty("Mdp",Compte_c.get_Mdp());

                           SoapSerializationEnvelope envelope = new
SoapSerializationEnvelope(SoapEnvelope.VER11);
                           envelope.setOutputSoapObject(request);
                           AndroidHttpTransport androidHttpTransport = new
AndroidHttpTransport(URL);
                           androidHttpTransport.call(SOAP_ACTION, envelope);

                           SoapObject result = (SoapObject) envelope.getResponse();
                           Connexion_c.Session(

Integer.parseInt(result.getProperty("ID_Fam").toString()),

Boolean.parseBoolean(result.getProperty("Role").toString()));

                           NumErreur  =
Integer.parseInt(result.getProperty("Erreur").toString());
                         } catch (Exception e) {NumErreur=-1;}
              fin_Thread=true;
           }
         }).start();
        Synchro(3);
       return NumErreur;
}
```

a.2 Requête SOAP :

Tout d'abord on fait appel à une méthode privée « init_liaison() » qui permet de réinitialiser le numErreur. Ensuite on construit, dans un thread secondaire, la requête SOAP, pour cela on utilise la bibliothèque **KSOAP2.** Tout d'abord on spécifie le **NAMESPACE** *(équivalent à l'URL du serveur)* et on ajoute le nom de la méthode que l'on souhaite utiliser, ici: «SeConnecter»
Le **WSDL**, *cf. ANNEXE,* spécifie que cette méthode a besoin de recevoir 2 propriétés à savoir l'email et le mdp.

```
- <xs:complexType name="SeConnecter">
  - <xs:sequence>
      <xs:element minOccurs="0" name="Email" type="xs:string"/>
      <xs:element minOccurs="0" name="Mdp" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
```

On ajoute donc ces 2 champs à notre requête.
En ce qui concerne la ligne *AndroidHttpTransport*, on indique **l'URL** du WSDL situé sur le serveur,

puis pour la méthode *call* on appel le **SOAP_ACTION** du webservice que l'on souhaite utilisé.
Dans notre cas voici à quoi correspondent les variables NAMESPACE, URL, et SOAP_ACTION :

```
private String NAMESPACE = "http://192.168.3.5:8080/Actions_Client";
private String URL = "http://192.168.3.5:8080/E-periscol/Actions_Client?WSDL";
private String SOAP_ACTION = "http://192.168.3.5:8080/E-periscol/Actions_Client";
```

Pour finir, on reçoit la réponse du serveur via la méthode enveloppe.getResponse();
On traite cette réponse en récupérant les valeurs des différentes propriétés contenu dans le message XML de la réponse. A savoir, l'ID_Famille, le rôle *(admin ou user)*, et le numéro d'erreur.

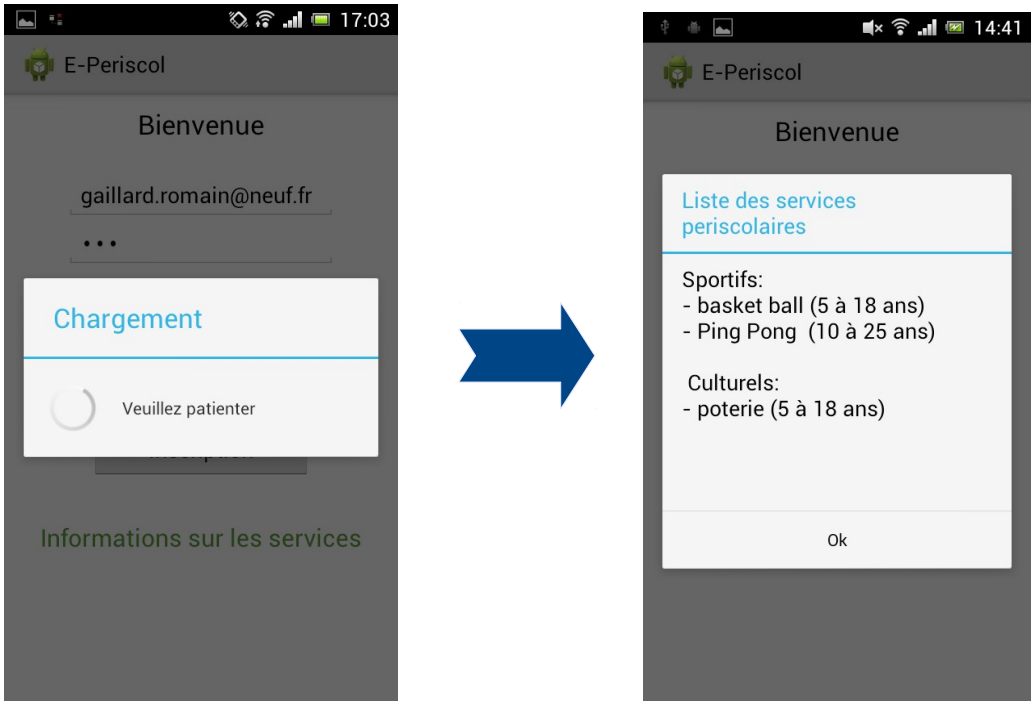
On signal ensuite la fin du Thread avec une variable boolean. Pendant ce temps là, dans le thread principal on a fait appel à la méhode **Synchro()** qui permet de mettre en attente le thread principal pour attendre que la requête SOAP se termine.

a.3 La méthode Synchro

```
private void Synchro(int duree){
      int i=0;
      while(fin_Thread!=true && i<=duree){try {Thread.sleep(300);} catch
(InterruptedException e) {e.printStackTrace();} i++;}
      };
```

La méthode Synchro consiste à endormir le Thread principal avec la méthode s*leep(),* ici on effectue un sleep de 300ms tant que le Thread secondaire n'indique pas qu'il a terminé, via la variable *fin_Thread*. A noté qu'on peut spécifier une durée qui correspond au nombre de boucle maximum qui sera effectuée dans la méthode. Ainsi on peut considérer qu'au bout de 3 ou 5 tours de boucle, il est inutile de continuer à attendre le Thread secondaire.

a.4 Consulter les services



Lors de l'appui sur le bouton «Information sur les services» présent sur l'IHM de Connexion, la

méthode get_AllService() du client_cli est lancée.

```
case R.id.tvInfoService:
       myProgressDialog = ProgressDialog.show(IHM_Connexion.this,"Chargement", "Veuillez
patienter", true);
       new Thread(new Runnable() {
           public void run() {
               result=false;
               result=client.get_AllService();
               myProgressDialog.dismiss();
              Message msg = mHandler.obtainMessage(MSG_SERVICES);
               mHandler.sendMessage(msg);
               }
           }).start();
       break;
```

On procède de la même manière que pour lancer la méthode *Connect* précédemment étudié.
A noté que cette fois-ci on envoi aucun paramètre dans la requête SOAP et on récupère un catalogue de service.
Code de la méthode get_AllService() de la classe Client_cli :

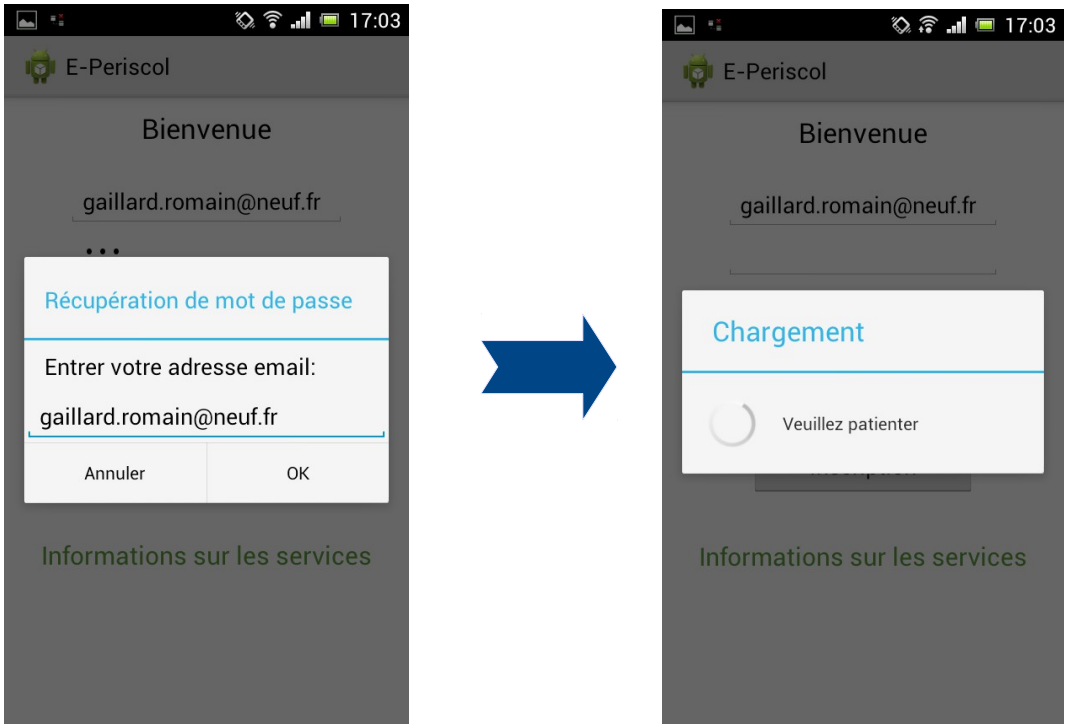
```
SoapObject request = new SoapObject(NAMESPACE,"Get_All_Service");
SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
envelope.setOutputSoapObject(request);
AndroidHttpTransport androidHttpTransport = new AndroidHttpTransport(URL);

androidHttpTransport.call(SOAP_ACTION, envelope);

SoapObject result  = (SoapObject) envelope.getResponse();
CataloguesServices_c.Delete() ;
// Récupérer tableau de serv
for(int i=0;i<result.getPropertyCount();i++){
       SoapObject resultServices = (SoapObject) result.getProperty(i);
       CataloguesServices_c.Ajouter(resultServices.getProperty("Nom").toString(),
                                    resultServices.getProperty("Domaine").toString(),
              Short.parseShort(resultServices.getProperty("AgeMin").toString()),
              Short.parseShort(resultServices.getProperty("AgeMax").toString()));
}
```

Explications: Au niveau de la réception, on reçoit un catalogue composé de plusieurs objets services dont la structure est définie dans le WSDL *(cf. Annexe)*
On compte le nombre d'objet « Service » reçu et on réalise une boucle, afin de remplir notre catalogue en local auquel on ajoute chaque service reçu.
Zoom sur la classe Catalogue:

```
private ArrayList<Service> Services;
public CatalogueServices(){  Services = new ArrayList<Service>(); }
public void Ajouter(String nom,String domaine,Short ageMin,Short ageMax){
       Service temp = new Service();
       temp.Ajouter(nom,domaine,ageMin,ageMax);
       Services.add(temp);
       }
```

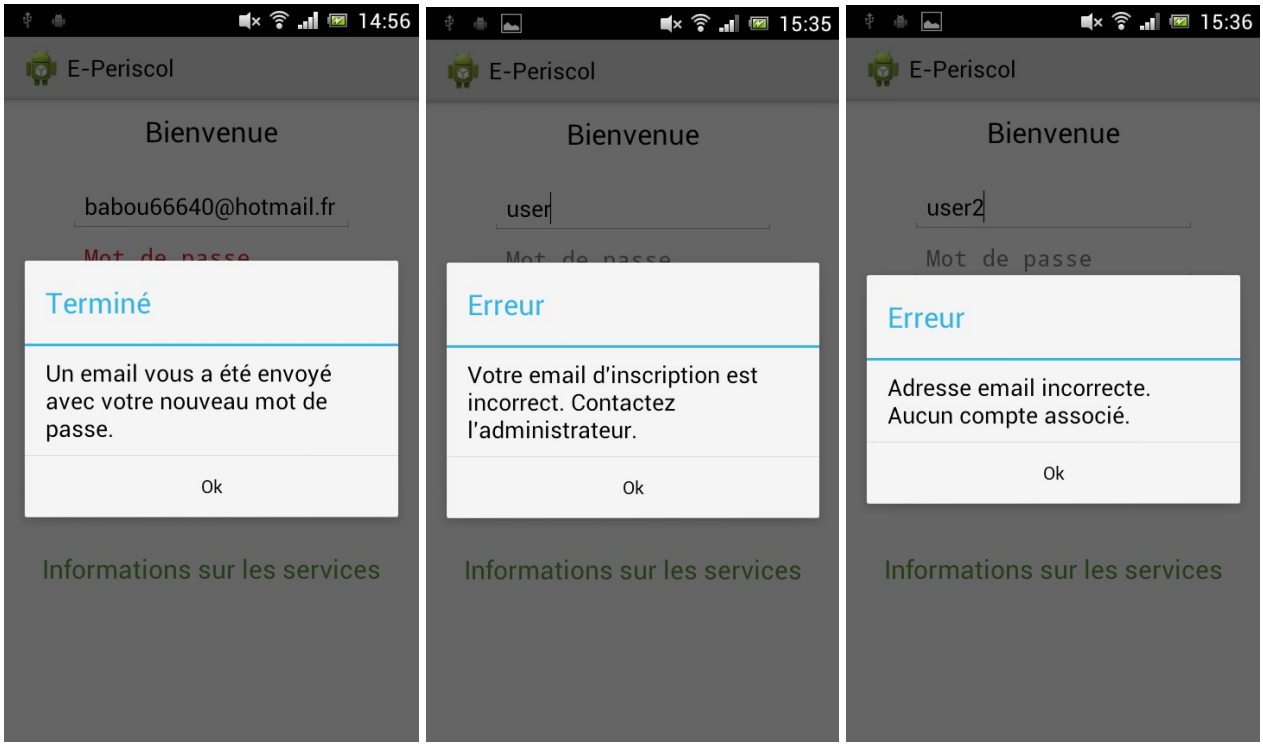
On créé un tableau dynamique *(ArrayList)* de service à la création du catalogue. Ensuite la méthode Ajouter permet de créer un nouveau Service et d'ajouter ce service à notre tableau de service.

a.5 Récupération de mot de passe

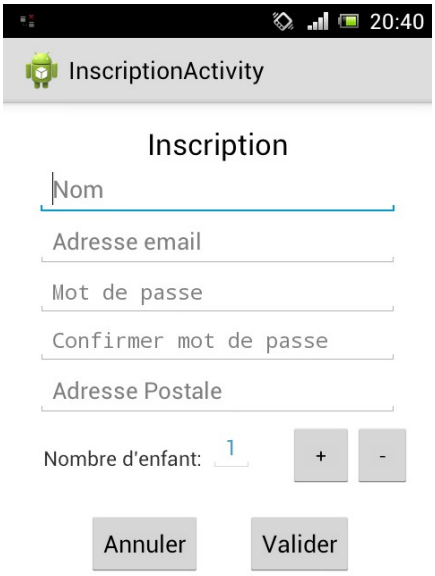


Lors de l'appui sur le bouton '**Mot de passe oublié**' de l'IHM Connexion, la popup ci-dessus s'affiche. Elle possède un editText où l'utilisateur peut entrer son adresse email. A noté que ce composant est pré-rempli avec la valeur du champ email de l'IHM Connexion, si celui-ci était rempli. L'appui sur le bouton '**Ok**' déclenche la méthode *mdp_oublie()* de la classe Client_cli, elle même créée la requête SOAP en utilisant la méthode MdpOublie du Web Service.
Le code de fonctionnement est similaire à la méthode connexion, vu précédemment. On envoi en paramètre l'email, et le serveur se charge de vérifier que ce mail est dans la BDD, si c'est le cas il envoi un email à l'adresse indiquée. En cas d'adresse erronée le serveur renvoi une erreur.

Voici les résultats possibles pour ce cas d'utilisation :



b. IHM Inscription

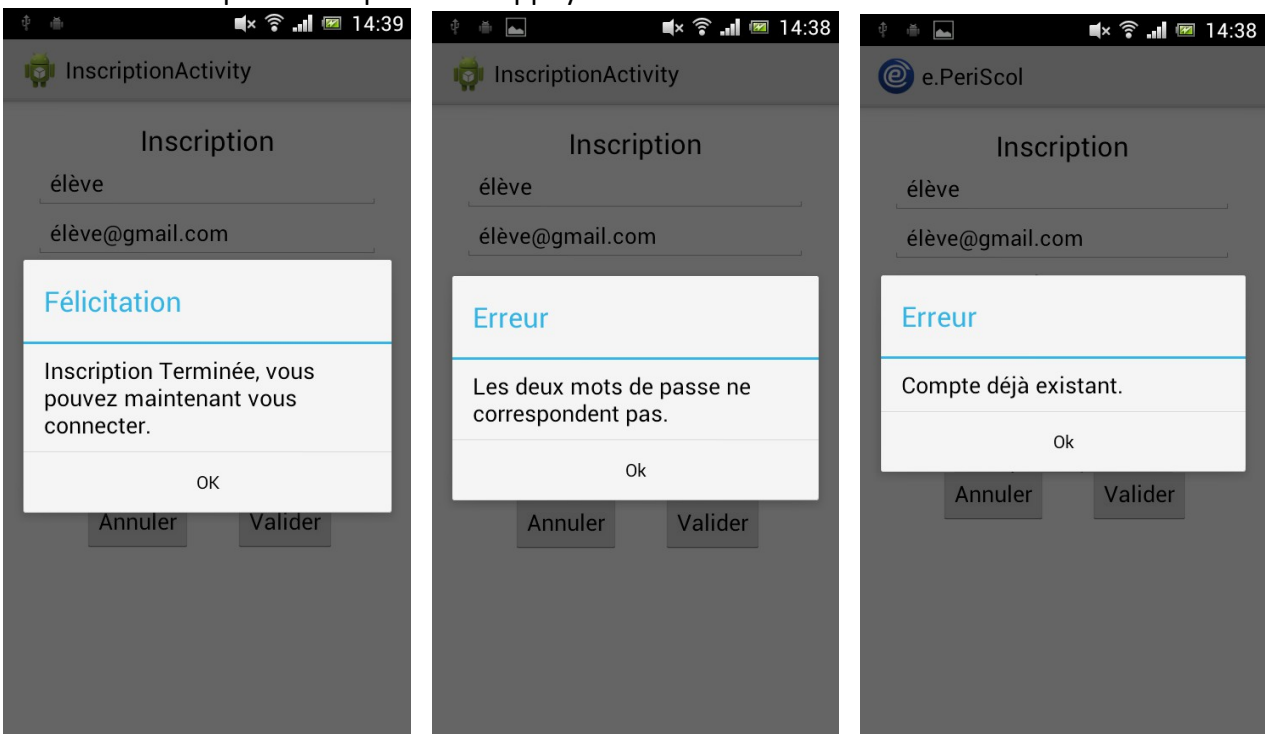


Cette IHM permet de se créer un compte sur l'application e-PeriScol. A noter que le nombre d'enfant se configure à l'aide des boutons '**+**' et '-' situé à droite du champ.
Le bouton '**Annuler**' permet de revenir à l'IHM Connexion.
Le bouton '**Valider**' exécute la méthode *set_Compte()* de la classe Client_cli pour envoyer la

requête SOAP vers le serveur Jboss en utilisant la méthode *CreerCompte()* du Web Service.
Le fonctionnement du code est similaire à celui utiliser pour la connexion.
→ Cf. section précédente.

Par ailleurs, on effectue une vérification sur la correspondance des deux mots de passe, ainsi que sur le format de l'adresse email.

Voici les résultats possibles après avoir appuyer sur le bouton Valider:



Il peut également y avoir une erreur indiquant que le **format de l'adresse email** est incorrecte. Cependant, cette fonction n'est pas réalisée pour le moment.

De même, on peut retrouver **l'erreur de connexion** au serveur. → *Cf. screen de la connexion section VIII. 4) a)*

Nous avons choisi de réaliser une inscription rapide pour l'utilisateur. Par conséquent, nous n'avons pas souhaiter mettre en œuvre un système de confirmation d'email.

c. IHM Accueil (Connecté)

Cet écran apparaît après la connexion de l'utilisateur :

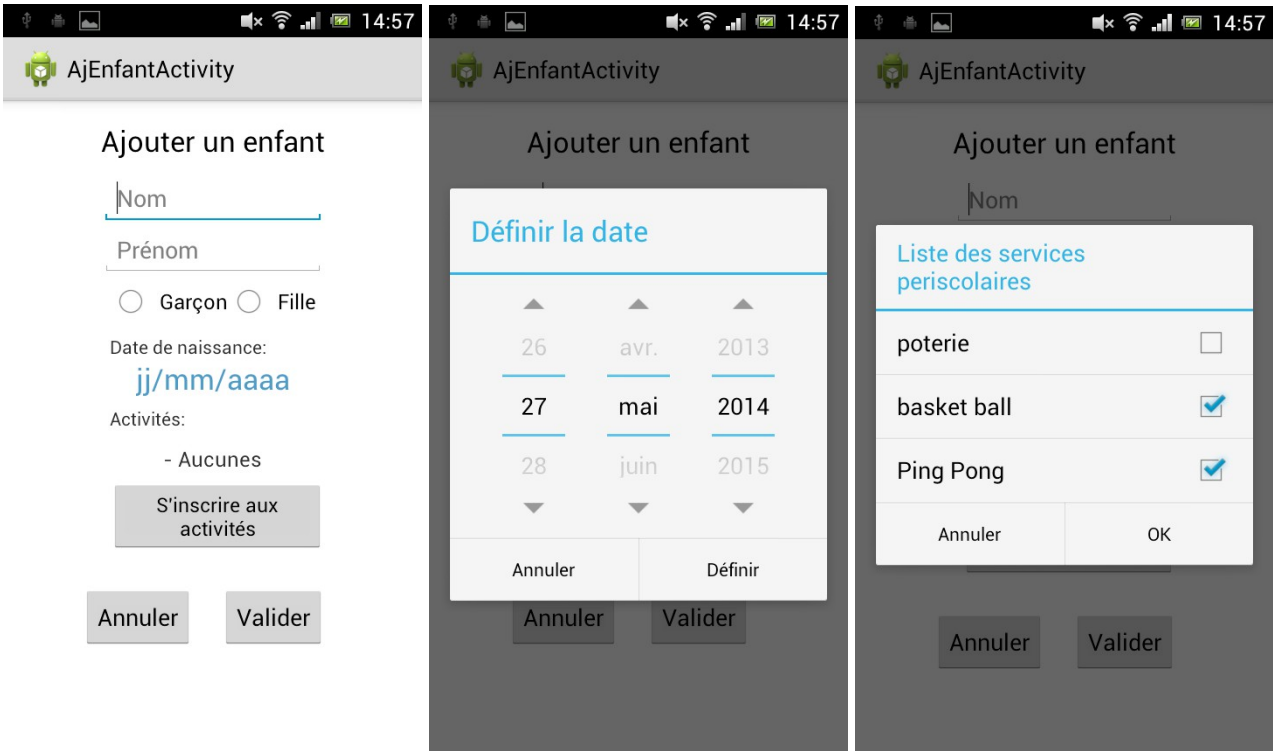
Il permet l'affichage des enfants qui sont reliés au compte, ainsi que la modification des services périscolaires auxquelles les enfants sont inscrits. Il est également possible de modifier les paramètres de l'enfant renseignés lors de son ajout au compte.
De plus, cette interface permet, via le bouton '**Ajouter Enfant**', d'accéder à l'*IHM Ajouter un Enfant*. Le bouton '**Paramètres**' redirige vers l'*IHM Modifier Compte* qui rend possible la modification des paramètres du compte e-Periscol (email, mot de passe, nombre d'enfant, etc.) entrés lors de l'inscription. Pour finir le bouton '**Déconnexion**' renvoi l'utilisateur à l'IHM de Connexion.

Lors du lancement de l'IHM d'Accueil *(connecté)* on exécute la méthode *get_Enfant()* de la classe Client_cli. Cette méthode nécessite un paramètre qui correspond à l'id_famille. La méthode construit la requête SOAP à destination de la méthode *Get_Enfant()* du serveur de Web Services. Grâce à l'ID_famille le serveur est en mesure de nous renvoyer la liste des enfants.

A noté qu'à la réception des enfants, on génère dynamiquement les composants qui permettent d'afficher les informations sur les enfants associés au compte. On génère donc autant de composant qu'il y a d'enfant. → *cf. Annexe pour voir le code de l'IHM_Accueil.java*

d. IHM Enfant (Ajouter)

Cet écran apparaît après avoir cliquer sur '**Ajouter Enfant**' sur l'IHM d'Accueil :



Un formulaire est à remplir pour renseigner les informations personnelles de l'enfant.
Concernant la date de naissance, j'ai opté pour le choix d'une popup DatePickerDialog pour améliorer l'ergonomie et éviter ainsi à l'utilisateur d'entrer lui même la date avec le clavier.
Dans un même temps, l'interface propose également d'inscrire l'enfant aux services périscolaires proposées, via le bouton '**S'inscrire aux activités**'. Ce bouton déclenche la popup qui affiche la liste des activités disponibles en fonction de l'âge de l'enfant. Pour cela, la méthode get_AllService() du client_cli est lancée. → *cf. section VIII. 4) a.4*
Extrait du code du fonctionnement de la popup comprenant la liste des checkBox :

```
Recup_Activites(); // On récupère les activités disponible en fonction de la date de naissance.
final ArrayList<Integer> mSelectedItems; // Liste des numéro d'activités sélectionnées.
mSelectedItems = new ArrayList<Integer>();
CharSequence[] items = new String[Activites.size()]; // tableau des noms des activités
for(int j=0;j<Activites.size();j++){
	items[j]=Activites.get(j).get_Nom(); // On récupère le nom des # services
}

MultiChoiceClickListener = new DialogInterface.OnMultiChoiceClickListener() { // composant
MultiChexbox pour la popup
@Override
public void onClick(DialogInterface dialog, int indexSelected,boolean isChecked) {
	if (isChecked) {
	mSelectedItems.add(indexSelected);
	mCheckedItems[indexSelected]=true; // etat de la checkbox
	} else if (mSelectedItems.contains(indexSelected)) {
	mSelectedItems.remove(Integer.valueOf(indexSelected));
	mCheckedItems[indexSelected]=false;
	}
 }
};
final boolean[] mCheckedItems_save = new boolean[Activites.size()];
for(int i=0;i<Activites.size();i++)
	mCheckedItems_save[i]=mCheckedItems[i]; // sauvegarde de l'état de chaque chexbox
AlertDialog.Builder builder = new AlertDialog.Builder(IHM_Enfant.this);
// construction de la popup
builder.setTitle("Liste des services periscolaires");
builder.setMultiChoiceItems(items,mCheckedItems,MultiChoiceClickListener);
builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
	public void onClick(DialogInterface dialog, int id) {}
	});
builder.setNegativeButton("Annuler", new DialogInterface.OnClickListener() {
	@Override
	public void onClick(DialogInterface dialog, int id) {
	// Restauration
	for(int i=0;i<Activites.size();i++)
		MCheckedItems[i]=mCheckedItems_save[i];
	}
});
AlertDialog dialog = builder.create();
dialog.show();
```

Enfin, le bouton '**Annuler**' permet de retourner à l'IHM Accueil.
Et le bouton '**Valider**' exécute la méthode *set_Enfant()* de la classe Client_cli qui envoi la requête SOAP vers le serveur via la méthode *AjouterEnfant()* de celui-ci. → *cf. Section VIII. 4) a.*

Les erreurs possibles pour cette IHM sont :

- Erreur de connexion au serveur
- Il n'y a plus de place pour s'inscrire à *nom_activité.*
- Félicitation, votre enfant a bien été ajouté à votre compte. Il est inscrit au(x) service(s):
  liste_des_services

e. IHM Modifier Enfant

Cette IHM et son fonctionnement sont similaires à l'IHM Ajouter Enfant. → *cf. Section VIII. 4) d.*
A l'exception que les champs sont pré-rempli avec les informations de l'enfant, et qu'il y a un bouton '**Supprimer**' pour retirer l'enfant du compte.

f. IHM Paramètres (du Compte)

Cette IHM et son fonctionnement sont similaires à l'IHM Inscription. → *cf. Section VIII. 4) d.*
A l'exception, des champs mot de passe. Qui sont remplacé par un bouton '**Changer le mot de passe**' qui ouvre une popup avec les champs : 'Mot de passe actuelle', 'Nouveau mot de passe', 'Confirmer nouveau mot de passe'.

h. Bonus

h.1 Les bases de la programmation Android

Tout d'abord il est important de savoir que toutes les interfaces sont décrites dans un fichier **XML**. *(→ cf. Annexes activity_inscription.xml, etc.)*
Ce fichier peut être généré manuellement, ou automatiquement via l'assitant d'éclipse après avoir installé le SDK. → *cf. Dossier de Mise en Oeuvre partie Android.*

Chaque composant créé sur l'interface comporte un **ID** cet identifiant est très important, puisqu'il nous permet ensuite de gérer le composant dans notre code en java.
C'est ainsi qu'on procède pour récupérer dans notre code les vues de nos composants :

```
private Button btConnexion = null;
private EditText etEmail = null;
private TextView tvMdpForget = null;

protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

// Récupération des vues:
btConnexion = (Button)findViewById(R.id.btConnexion);
etEmail = (EditText)findViewById(R.id.etEmail);
tvMdpForget = (TextView)findViewById(R.id.tvMdpForget);
```

Ainsi de cette façon, on peut maintenant agir sur nos composants, en récupérant par exemple le text d'une EditText via la méthode *.getText()* ou modifer la valeur du text via *.setText(),* mais aussi la couleur des composants grâce à *.setTextColor(), et cetera.*

En ce qui concerne les boutons, il est nécessaire d'ajouter un listener sur ces composants afin de pouvoir gérer les événements qui s'y déroule. Le clic, le toucher, etc.
Exemple :

```
etEmail.setOnClickListener(button1Listener);
etEmail.setOnTouchListener(touch1Listener);
```

Ensuite lors déclenchement d'un événement, par exemple le click la méthode OnClick de la classe OnClickListener sera appelée et exécutera le traitement qu'on lui aura demandé d'effectuer. → cf. Annexes traitement des boutons *(IHM_Connexion par exemple)*
Il s'agit de classe issue de bibliothèques qu'il faut importer :

```
import android.view.View.OnClickListener;
import android.view.View.OnTouchListener;
```

h.2 Lancer une nouvelle activité

Pour changer d'IHM et afficher une nouvelle interfaces à l'écran, on utilise le code suivant :
Exemple pour lancer l'IHM **Inscription** depuis l'IHM de **Connexion**.

```
Intent intent_Inscription = new Intent(IHM_Connexion.this,IHM_Inscription.class);
startActivity(intent_Inscription);
```

h.3 Passage de paramètres entre activités

Il peut être utile de transmettre certaine variable entres IHM. Par exemple, lors de la connexion l'IHM Connexion récupère un ID Famille, en cas de succès, via la classe Connexion_cli utilisé par la classe Client_cli. Dans notre cas l'ID Famille est une donnée importante. L'IHM d'Accueil (connecté) doit être en mesure d'afficher la liste des enfants correspondant au compte, donc à l'Id famille associé.
Pour passer cette variable, on ajoute la ligne suivante avant de lancer l'activité IHM Accueil.
Extrait du code de IHM Connexion:

```
intent_Accueil.putExtra("IdFam",Integer.toString(client.Connexion_c.get_
IdFam()));
```

Explications : Le premier paramètre indique le nom de la variable qu'on souhaite partager. Le deuxième paramètre indique sa valeur. Ici on va chercher la valeur dans un objet Connexion_c situé à l'intérieur de notre objet Client. → *cf. Diagramme de classe détaillé. Section VIII. 3) b.*

Par ailleurs, on récupère dans l'IHM Accueil cette valeur via le code :

```
Int idFam;
Intent intent = getIntent(); // Récupération de l'IDFAM.
if (intent != null)
            idFam = Integer.parseInt(intent.getStringExtra("IdFam"));
```

h.4 Le scrolling

Par défaut lorsqu'on créé une interface, elle est fixe et on ne peut pas la faire défiler lorsqu'elle occupe une place supérieure à la taille de l'écran. Pour résoudre ce problème il faut encadrer, dans le fichier xml qui décrit l'interface, les composants que l'on souhaite inclure dans le scrolling par la balise:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/ScrollView01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

// Description XML des différents composants

</ScrollView>
```

6) Test de validation

a. Tests techniques

Afin de valider le bon fonctionnement de la communication entre mon application et le serveur Jboss, nous avons utilisé **WireShark**, logiciel qui permet d'analyser les paquets réseaux.
Ce logiciel a été très utile pour le test et la mise au point, afin de déterminer qui du serveur ou du client était responsable.
J'ai choisi de montrer 3 captures WireShark concernant les cas d'utilisation Connexion, Inscription, et Consulter les services. Les captures étant relativement similaires.

a. 1 Connexion

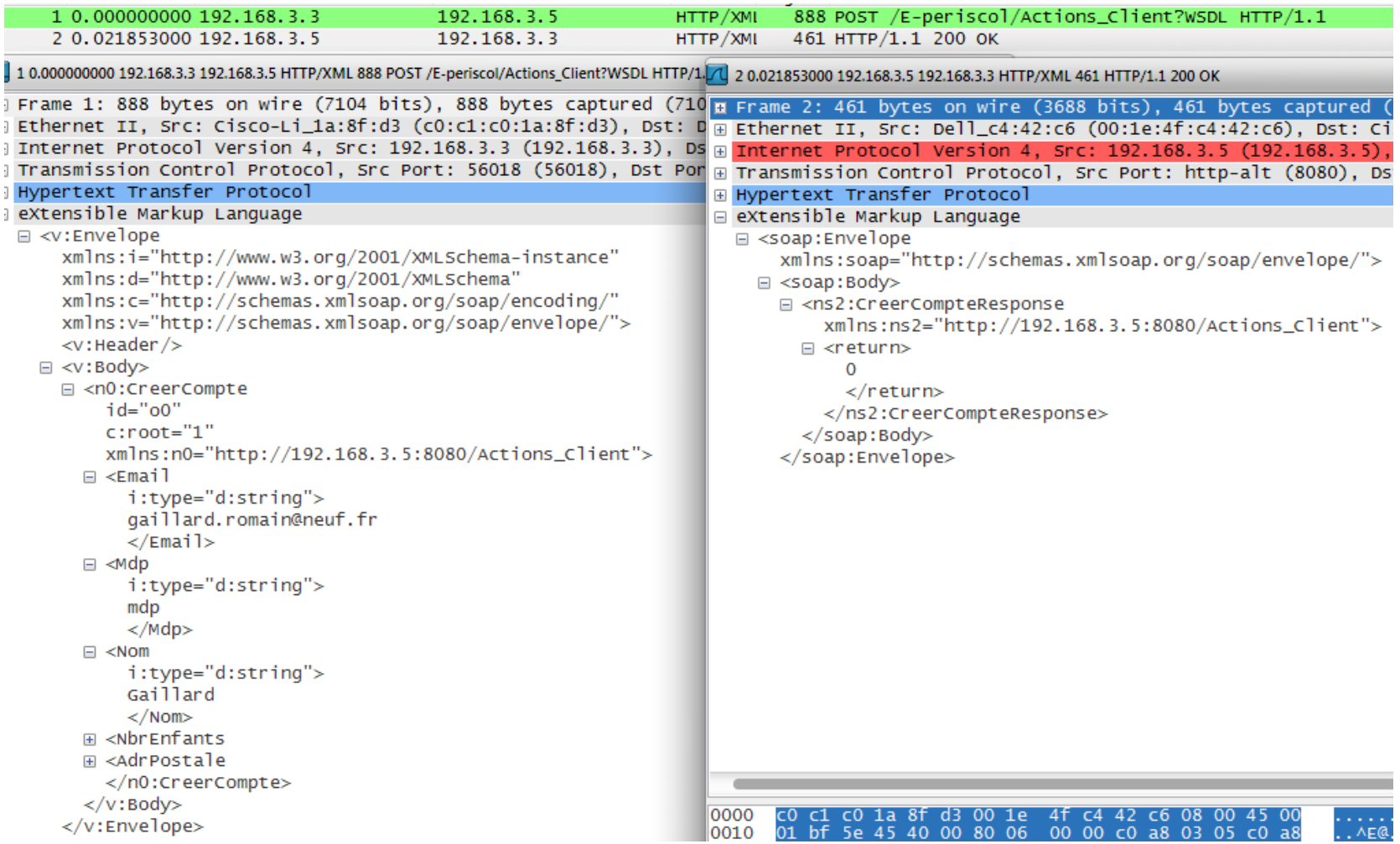
```
    1 0.000000000 192.168.3.17    192.168.3.13    HTTP/XMI   741 POST /E-periscol/Actions_Client?WSDL HTTP/1.1
    2 0.017964000 192.168.3.13    192.168.3.17    HTTP/XMI   514 HTTP/1.1 200 OK
```

```
⊞ Frame 1: 741 bytes on wire (5928 bits), 741 bytes captured (5928 bits) on interface 0
⊞ Ethernet II, Src: Cisco-Li_1a:8f:d3 (c0:c1:c0:1a:8f:d3), Dst: Dell_c4:42:c6 (00:1e:4f:c4:
⊞ Internet Protocol Version 4, Src: 192.168.3.17 (192.168.3.17), Dst: 192.168.3.13 (192.168
⊞ Transmission Control Protocol, Src Port: 54873 (54873), Dst Port: http-alt (8080), Seq: 1
⊞ Hypertext Transfer Protocol
⊟ eXtensible Markup Language
  ⊟ <v:Envelope
      xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:d="http://www.w3.org/2001/XMLSchema"
      xmlns:c="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:v="http://schemas.xmlsoap.org/soap/envelope/">
      <v:Header/>
    ⊟ <v:Body>
      ⊟ <n0:SeConnecter
          id="o0"
          c:root="1"
          xmlns:n0="http://192.168.3.13:8080/Actions_Client">
        ⊟ <Email
            i:type="d:string">
            user
            </Email>
        ⊟ <Mdp
            i:type="d:string">
            mdp
            </Mdp>
          </n0:SeConnecter>
        </v:Body>
      </v:Envelope>
```

```
⊞ Frame 2: 514 bytes on wire (4112 bits), 514 bytes captured (4
⊞ Ethernet II, Src: Dell_c4:42:c6 (00:1e:4f:c4:42:c6), Dst: Cis
⊞ Internet Protocol Version 4, Src: 192.168.3.13 (192.168.3.13)
⊞ Transmission Control Protocol, Src Port: http-alt (8080), Dst
⊞ Hypertext Transfer Protocol
⊟ eXtensible Markup Language
  ⊟ <soap:Envelope
      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    ⊟ <soap:Body>
      ⊟ <ns2:SeConnecterResponse
          xmlns:ns2="http://192.168.3.13:8080/Actions_Client">
        ⊟ <return>
          ⊟ <Erreur>
              0
              </Erreur>
          ⊟ <ID_Fam>
              3
              </ID_Fam>
          ⊟ <Role>
              true
              </Role>
            </return>
          </ns2:SeConnecterResponse>
        </soap:Body>
      </soap:Envelope>
```

La requête du client (à gauche) contient deux champs: Email et Mdp de type String.
Le serveur renvoi une réponse (à droite) contenant trois champs: Erreur, ID_Fam, Role.
(Le code 0 pour l'erreur correspond à une réussite)

a. 2 Inscription



Le client envoi une requête (à gauche) contenant les informations de chaque champs de l'IHM Inscription.
Le serveur répond en renvoyant uniquement un simple int indiquant le numéro d'erreur. *(Le code 0 correspondant à une réussite)*

a. 3 Consulter les services

```
1 0.000000000 192.168.3.3   192.168.3.5   HTTP/XMI   658 POST /E-periscol/Actions_Client?WSDL HTTP/1.1
2 0.024563000 192.168.3.5   192.168.3.3   HTTP/XMI   837 HTTP/1.1 200 OK
```

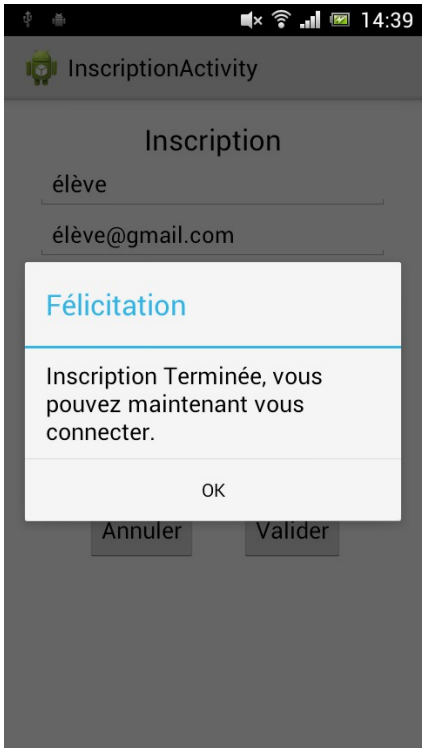
```
Frame 1: 658 bytes on wire (5264 bits), 658 bytes captured (52
Ethernet II, Src: Cisco-Li_1a:8f:d3 (c0:c1:c0:1a:8f:d3), Dst:
Internet Protocol Version 4, Src: 192.168.3.3 (192.168.3.3), D
Transmission Control Protocol, Src Port: 36232 (36232), Dst Po
Hypertext Transfer Protocol
eXtensible Markup Language
  <v:Envelope
      xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:d="http://www.w3.org/2001/XMLSchema"
      xmlns:c="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:v="http://schemas.xmlsoap.org/soap/envelope/">
      <v:Header/>
    <v:Body>
      <n0:Get_All_Service
          id="o0"
          c:root="1"
          xmlns:n0="http://192.168.3.5:8080/Actions_Client"/>
      </v:Body>
    </v:Envelope>
```

```
Frame 2: 837 bytes on wire (6696 bits), 837 bytes captured (6696 bits) on int
Ethernet II, Src: Dell_c4:42:c6 (00:1e:4f:c4:42:c6), Dst: Cisco-Li_1a:8f:d3 (
Internet Protocol Version 4, Src: 192.168.3.5 (192.168.3.5), Dst: 192.168.3.3
Transmission Control Protocol, Src Port: http-alt (8080), Dst Port: 36232 (36
Hypertext Transfer Protocol
eXtensible Markup Language
  <soap:Envelope
      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
      <ns2:Get_All_ServiceResponse
          xmlns:ns2="http://192.168.3.5:8080/Actions_Client">
        <Catalogue>
          <Service>
            <Nom>
              poterie
              </Nom>
            <Domaine>
              Culturel
              </Domaine>
            <AgeMin>
              5
              </AgeMin>
            <AgeMax>
            <NbrMax>
            </Service>
          <Service>
            <Nom>
            <Domaine>
            <AgeMin>
            <AgeMax>
            <NbrMax>
            </Service>
          <Service>
          </Catalogue>
      </ns2:Get_All_ServiceResponse>
```

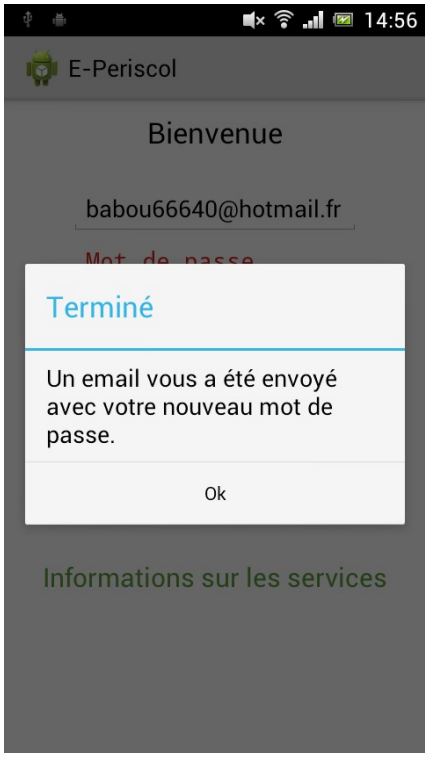
Ici le client, envoi aucun paramètre. En revanche, le serveur lui renvoi un tableau d'objet service. Il y a 3 services, composés chacun d'un nom, d'un domaine, d'un age minimum et maximum, ainsi que d'un nombre maximum de participant à l'activité.

b. Test fonctionnels

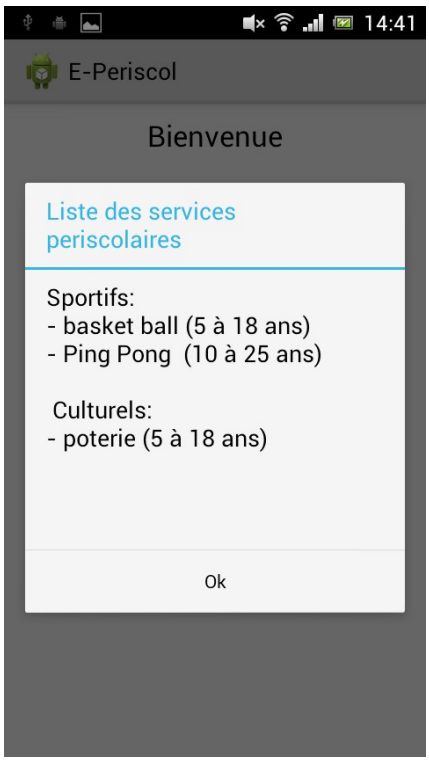
En ce qui concerne les tests fonctionnels, vous pouvez vous référer aux tests fonctionnels de **Baptiste Cahuzière** et **Dorian Caup** qui sont rendu possible grâce à mon application.
Par ailleurs, en ce qui me concerne, voici les résultats positifs des différents cas d'utilisation qu'il existe:



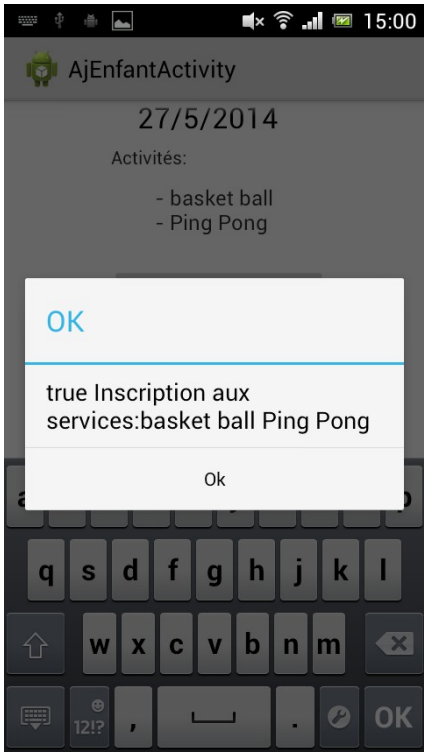
Inscription



Mot de passe oublié



Consulter les services



Ajouter un Enfant

*Remarque : Le message affiché n'est pas dans sa version finale. Il s'agit d'une version de « débogage » qui indique le résultat du serveur, ici **'true'** pour indiquer le bon fonctionnement. Ainsi qu'un récapitulatif de la liste des services auxquelles l'enfant s'est inscrit.*

Au sujet de la connexion, il n'y a pas de popup visible puisque en cas de succès l'application nous redirige directement sur l'IHM d'Accueil.

7) Bilan personnel

Je suis très satisfait de ce projet pour plusieurs raisons.
D'une part, e-PeriScol est un projet concret, moderne et complet. Il est concret car son objectif concerne le milieu scolaire. En tant qu'étudiant cet environnement nous est familier, par conséquent la mise en situation est facile. Ainsi on comprend aisément l'intérêt et l'objectif du projet. D'autant que celui-ci est moderne, il s'inscrit dans un mouvement durable d'expansion des technologies mobiles. De plus, il est complet grâce à la diversité des aspects qui y sont abordés, à savoir Android, iOs, JavaFX2, MySQL, Jboss et les Web Services (SOAP).

D'autre part, e-PeriScol est un projet enrichissant en terme de connaissance. Il m'a permis de renforcer ma compréhension du langage Java et de la programmation orientée objet. Mais aussi de découvrir Android, et d'acquérir des méthodologies de codage de par la conception UML, ainsi que des bonnes pratiques de programmation.

Pour finir, e-PeriScol est un projet collectif dans lequel l'équipe a contribué à l'enrichissement de l'expérience de travail en commun. De par l'acquisition des méthodes de travail collectives, qui ont permis l'amélioration de ma notion de gestion de projet. Mais aussi sur le plan humain, grâce à une forte coopération et entraide au sein de l'équipe, dont l'ambiance a toujours été conviviale.

Je regrette de ne pas avoir pu finaliser mon application en la complétant avec l'ajout de photo pour chaque enfant et la création dynamique des composants d'affichage des enfants associé au compte. Avec plus de temps je serais en mesure de le faire.

IX. Partie personnelle de Cédric Cognard – Usager iOs:

1) Description de la partie personnelle

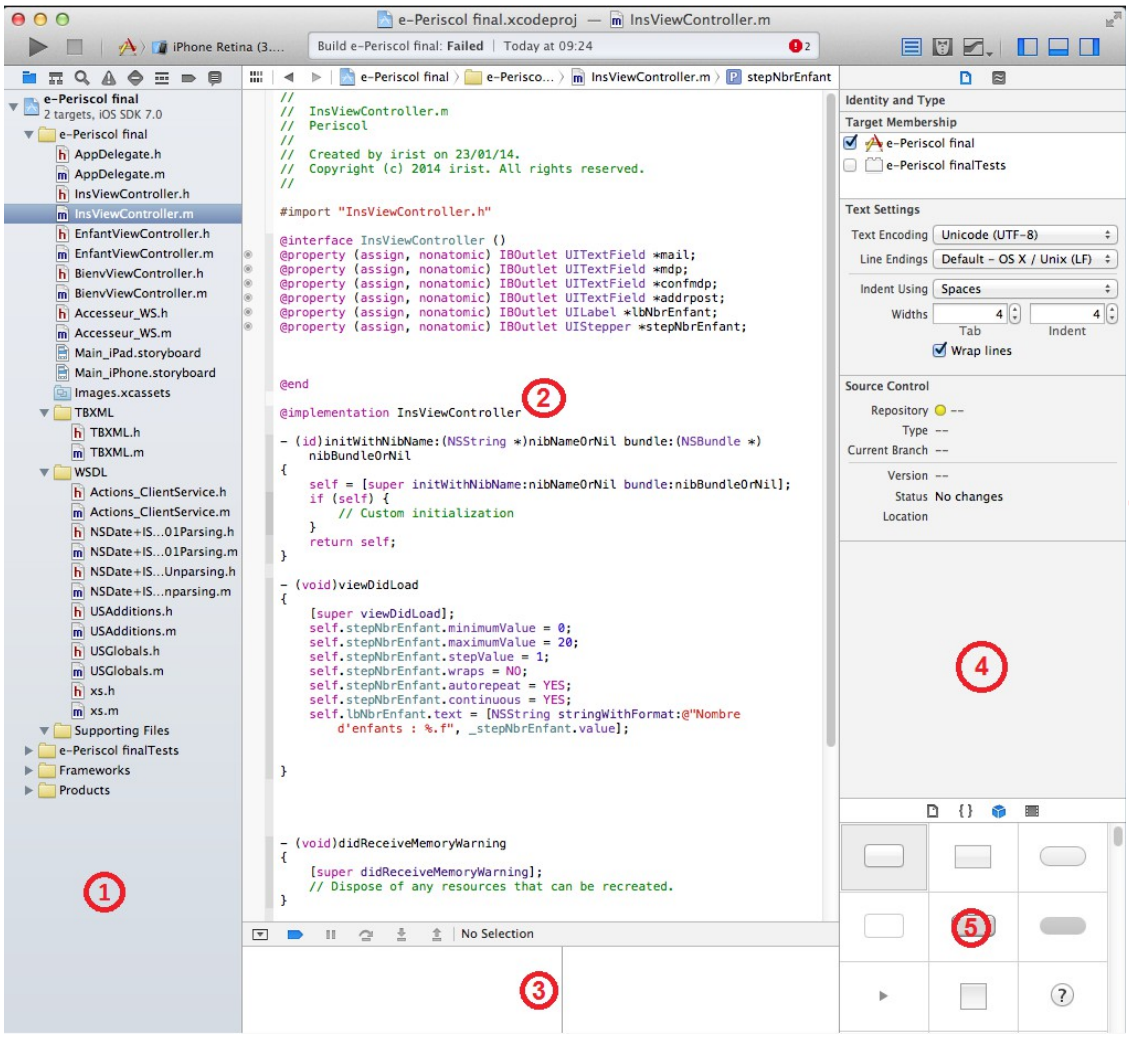
Pour pouvoir accéder au services proposés par une commune, les particuliers peuvent passer par l'application **e-PeriScol** sur **iPhone** pour inscrire ou désinscrire leurs enfants. Cette application est développée sous **iOS** qui fonctionne avec un affichage mono-vue**.**

2) Mise en œuvre :

a. Découverte Objective-C

Le système d'exploitation **iOS** est codé en **Objective-C** qui est un langage de programmation orienté objet. Chaque classe est composée des deux fichiers, un en-tête (.h) où l'on déclare les variables d'instance, les propriétés et les méthodes et le fichier implémentation (.m) où l'on code les méthodes.

b.Environnement de travail – Découverte de Xcode



1-

Arborescence des fichiers

2- Code du fichier en cours

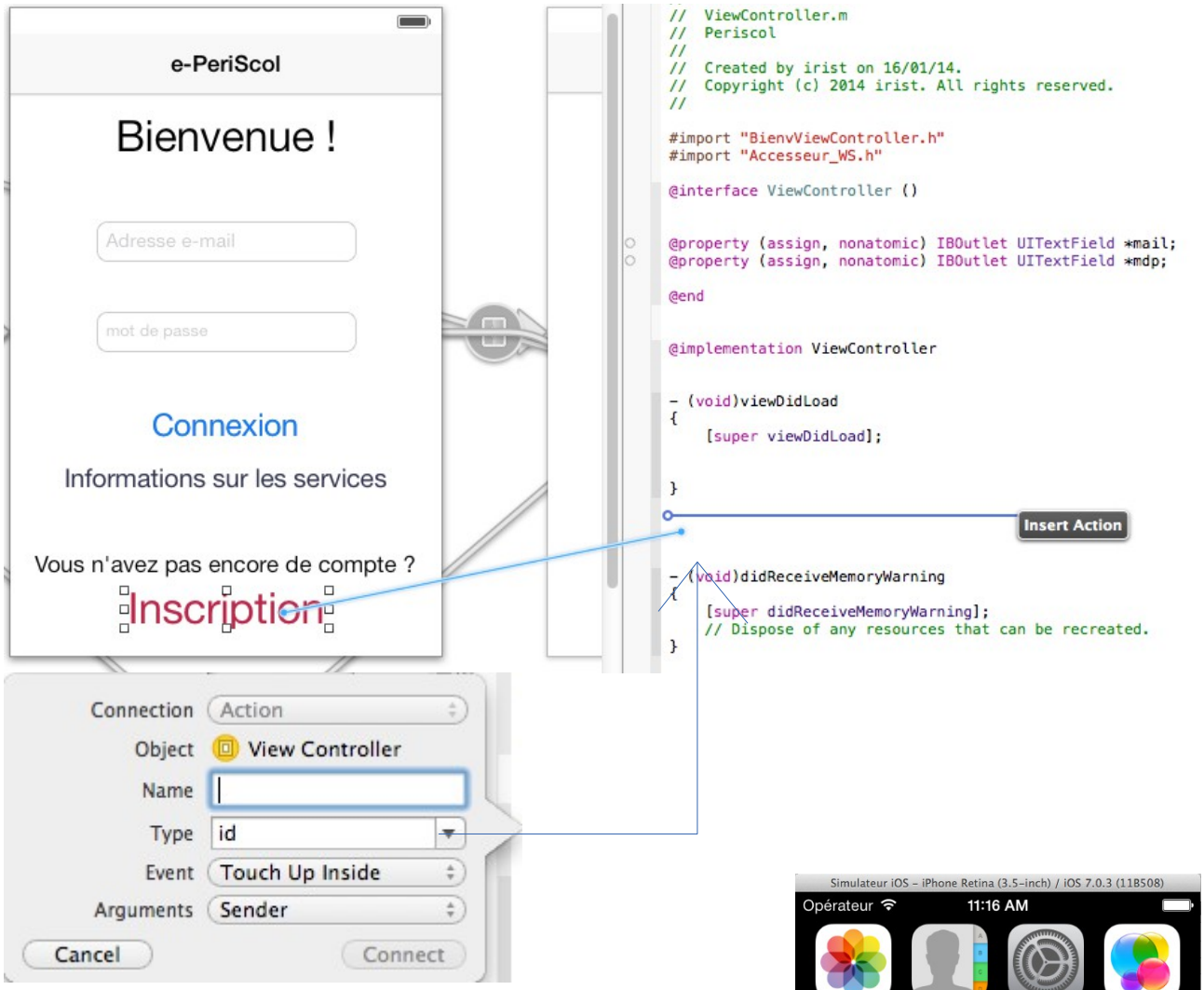
3- Console (affiche ce que fait le programme ou les erreurs)

4- Configuration des composants de l'IHM

5- Panel de composants pour l'IHM

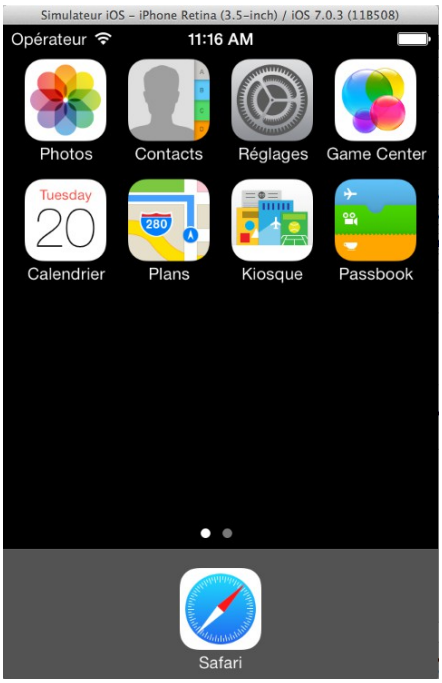
Cet environnement de développement permet de d'avoir une vue globale du projet et le codage des composants est simplifié par le fait de pouvoir de glisser vers le code un composant ce qui va avoir pour action d'ouvrir une fenêtre on l'on pourra définir quel type d'action on attend pour lancer la séquence de code. Ensuite cela nous créer le squelette du composant.

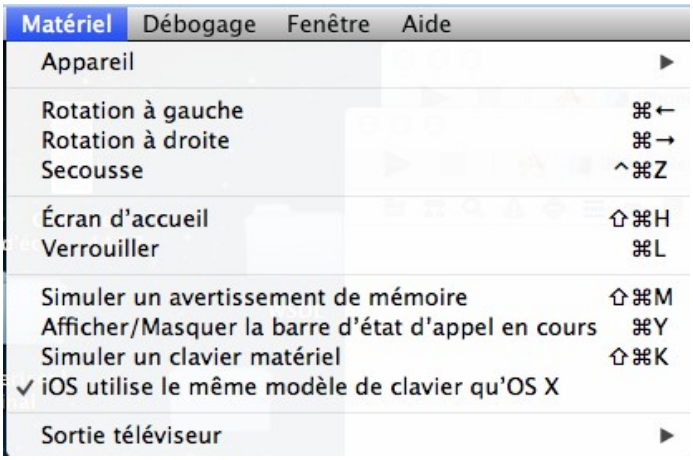
Exemple avec le bouton **Inscription** *(Création d'un nouveau compte)*:



c. Simulateur Xcode

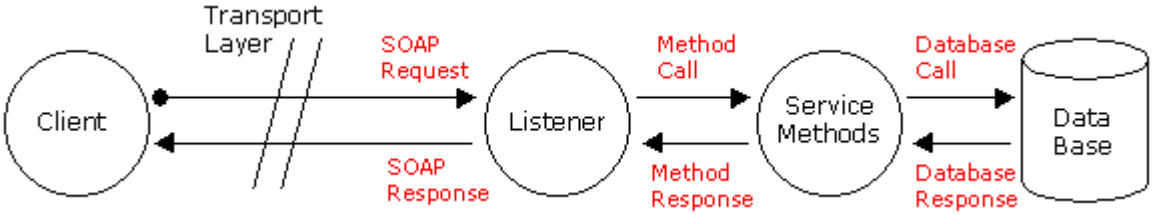
Le simulateur **iOS** fait partie intégrante de **Xcode**. Il permet de simuler le fonctionnement d'un **iPhone**, d'un **iPod Touch** et d'un **iPad** ce qui nous offre la possibilité de tester rapidement nos applications. On peut choisir le type de mobile, la version de l'**iOS** et simuler des actions comme par exemple le mobile sur le côté.





d. Web Service

Un web service est un programme informatique permettant la communication et l'échange de données entre une application client et un serveur via internet, ils communiquent entre eux par protocole **SOAP** (**Simple *Object Access Protocol***). Le serveur de web services et le client sérialisent et dé-sérialisent automatiquement les requêtes.



Lorsque le client appel le serveur **JBOSS**, il envoie un message **SOAP** c'est-à-dire un message en **XML**. Le serveur crée alors un **Listener** qui récupère les informations du **XML**, dé-sérialise les informations et appelle dynamiquement le web service demandé. Ce dernier traite les informations reçues par le client et les renvois au **Listener**. En retour, ce **Listener** sérialise les informations renvoyées et les renvois dans la réponse **SOAP**.

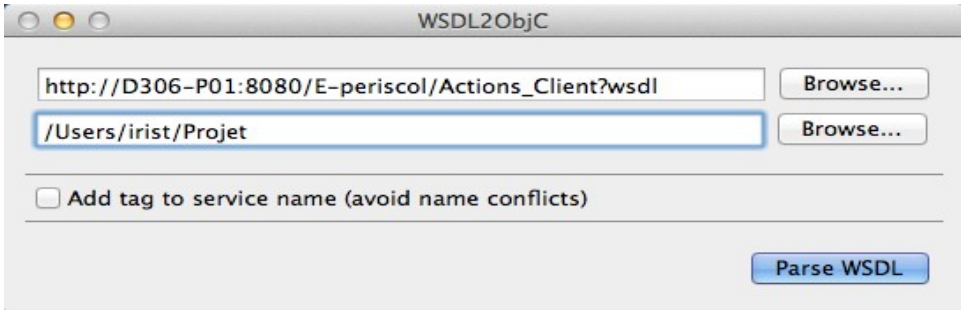
Pour l'utilisation d'un web service le fichier **wsdl** est très important. C'est lui qui contient toutes les informations sur le web service. Il donne les méthodes du serveur, le nom et le type de leurs paramètres et ce quelle retournent. (Voir annexe **WSDL**).

Par exemple, voici le **wsdl** de **SeConnecter, SeConnecterResponse** et **Connexion:**

```
<xs:complexType name="SeConnecter">
  - <xs:sequence>
      <xs:element name="Email" type="xs:string" minOccurs="0"/>
      <xs:element name="Mdp" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="SeConnecterResponse">
  - <xs:sequence>
      <xs:element name="return" type="tns:Connexion" minOccurs="0"/>
    </xs:sequence>
- <xs:complexType name="Connexion">
  - <xs:sequence>
      <xs:element name="Erreur" type="xs:int"/>
      <xs:element name="ID_Fam" type="xs:int"/>
      <xs:element name="Role" type="xs:boolean"/>
    </xs:sequence>
```

La méthode **SeConnnecter** a besoin qu'on lui passe deux paramètres, deux String : **Email** et **Mdp.** Sa réponse envoi un objet **Connexion** qui contient trois informations : **Erreur**, **ID_Fam** et **Role**.

Le problème que l'on rencontre sur **Xcode** est qu'il n'y a pas de librairie incluse pour utiliser le Web Service. On fait donc appelle à un logiciel qui va générer du code dans le projet à partir du fichier **WSDL**. Ce logiciel s'appelle **WSDL2ObjC**.



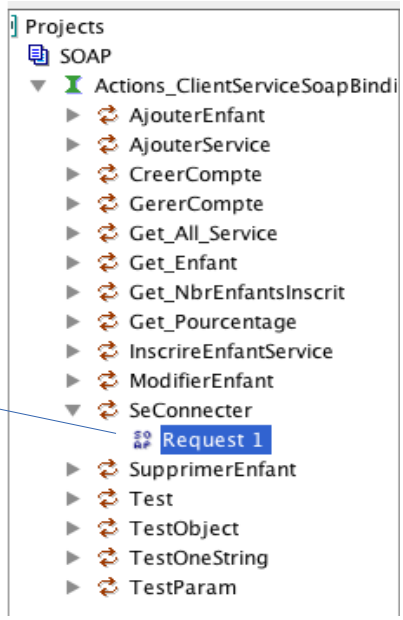
On lui donne l'adresse du **WSDL** et un emplacement de fichier pour qu'il y mette le code. Le logiciel génère ainsi plusieurs pages.

Ici **Actions_ClientService.h** et **Actions_ClientService.m** contiennent les méthodes du serveur. On peut donc désormais les appeler dans le projet.

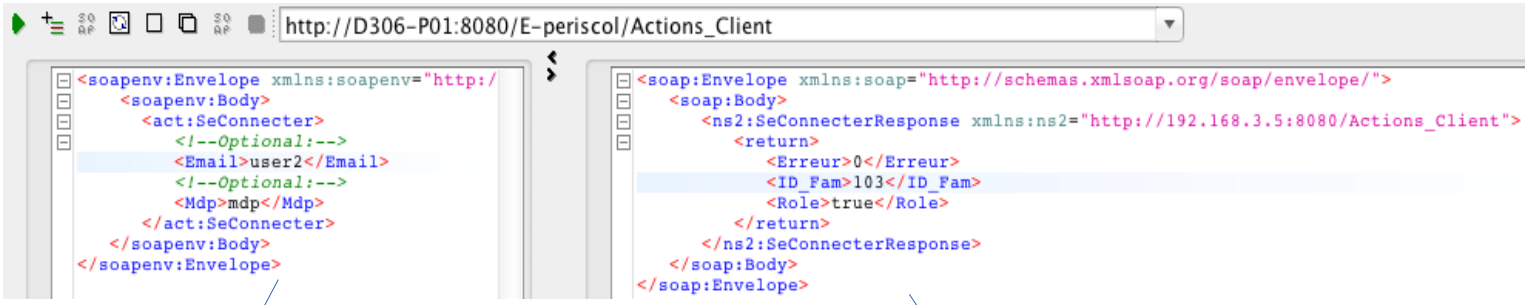
e. Aide à la mise en place du web service (**SoapUI**)

SoapUi est un logiciel qui permet de tester le web service, à partir du fichier **WSDL** il génère l'enveloppe **SOAP** que le serveur attend pour une requête sélectionnée, et il nous donne aussi l'enveloppe que le serveur renvoi. On peut ainsi comparer ce que notre programme envoi à l'aide de **Wireshark** et l'enveloppe type que le serveur attend, on peut donc corriger les erreurs.

Exemple avec **SeConnecter** :



Requête **SeConnecter**



Ce que le serveur attend

Ce que le serveur renvoi

3) Conception détaillée :

a. Diagramme de classes détaillé

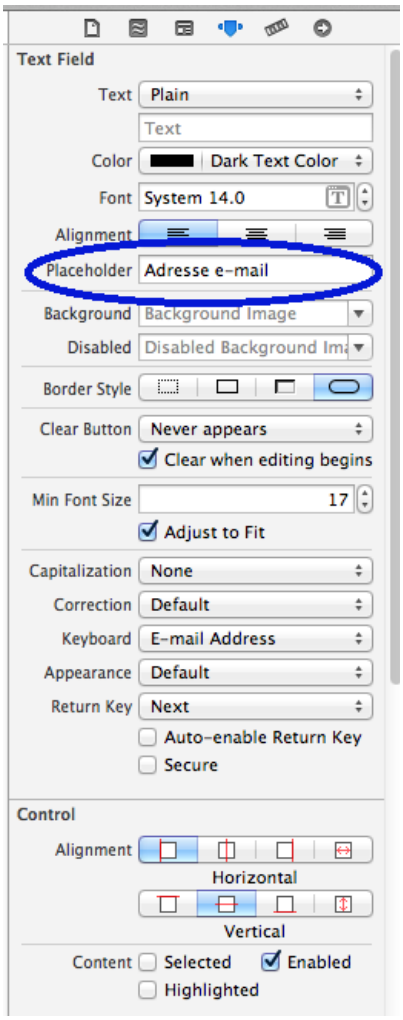
→ *cf. VIII. 3) b.*

b. Diagramme de séquences

→ *cf. VIII. 3) c. d. e. f. g. h. i. j. k. l.*

4) Implémentation :

a. Écran Bienvenue :



L'écran de bienvenue permet à l'utilisateur de se connecter ou créer un compte mais aussi d'afficher les services disponibles sans être connecter. Le bouton Connexion mène à la page **Accueil** si le compte existe. Le bouton Inscription mène à la page **Inscription.**

a.1. Champs textes :

Pour une question d'ergonomie, on souhaite afficher ce que l'application attend comme informations de la part de l'utilisateur. C'est pourquoi on affiche le contenu souhaité dans le composant comme ci-dessous.

En cliquant sur le composant

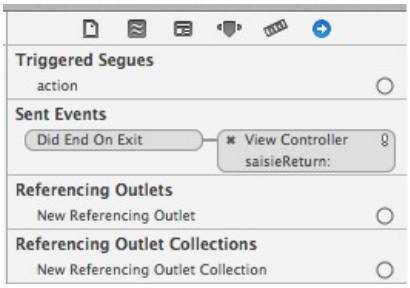
nous pouvons atteindre ses configuration et ainsi écrire «Adresse e-mail» qui sera en transparence et qui s'effacera dès que l'utilisateur appuiera sur un caractère.

Quand on appui sur un champ texte, le clavier s'affiche automatiquement, le problème que

l'on rencontre est que le bouton retour du clavier ne fait rien. Il faut donc coder une méthode dans l'implémentation (.m) qui cache le clavier.

```
-(IBAction)saisieReturn:(id)sender{
    [sender resignFirstResponder];
}
```

On assigne ensuite cette méthode au champ texte.



a.2. Connexion :

Pour se connecter à un compte utilisateur il faut s'authentifier avec un **email** et un **mot de passe**, si les données entrées sont celles d'un compte existant sur le serveur, alors la connexion peut se faire. La méthode **SeConnecter** du serveur renvoi un objet de type **Connexion** qui est composé d'**ID_Fam**, qui est un **int** qui identifie le compte. **NumErreur** qui est un **int** qui peut prendre 4 valeurs différentes (-1 pour un échec de connexion au serveur, 0 si il n'y a pas d'erreur, 1 en cas d'email inconnu, 2 en cas de mot de passe inconnu). Pour finir l'objet **Connexion** contient un **Booléen** qui vaut ***true*** pour un compte utilisateur et ***false*** pour un compte gestionnaire.

```
-(NSInteger) Connexion {
    Actions_ClientServiceSoapBinding *binding = [[Actions_ClientService
        Actions_ClientServiceSoapBinding] initWithAddress:@"http://192.168.3.5:8080/E-
        periscol/Actions_Client"];
    binding.logXMLInOut = YES;

    binding.authUsername = mail.text;
    binding.authPassword = mdp.text;

    Actions_ClientService_SeConnecter *requestSeConnecter =
        [[Actions_ClientService_SeConnecter alloc]init];

    requestSeConnecter.Email = binding.authUsername;
    requestSeConnecter.Mdp = binding.authPassword;



    Actions_ClientServiceSoapBindingResponse *respSeConnecter = [binding
        SeConnecterUsingParameters:requestSeConnecter];

     return 1;
}
```

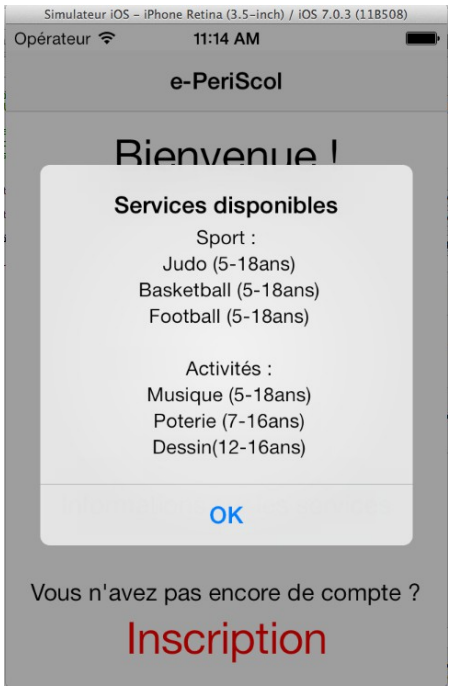
Passage des paramètres

Dans cette méthode on lie («**binding**») **Actions_ClientService** à l'**adresse** du serveur. On spécifie que l'on veut envoyer et recevoir des informations en **XML**. Les informations rentrées par l'utilisateur sont contenues dans «**mail.text**» et «**mdp.text**». On passe alors les paramètre dans la requête puis on spécifie que l'on souhaite un réponse en fonction des paramètres qu'on a passé dans la requête.

La méthode **Actions_ClientService_SeConnecter** est une méthode présente dans le code généré par **WSDL2ObjC** à partir du **WSDL** du serveur.

a.3. Affichage des services :

L'affichage des services se fait avec une Pop-up afin d'alléger l'application. Ici les données sont entrées en dure. On ferme la Pop-up en appuyant sur OK.



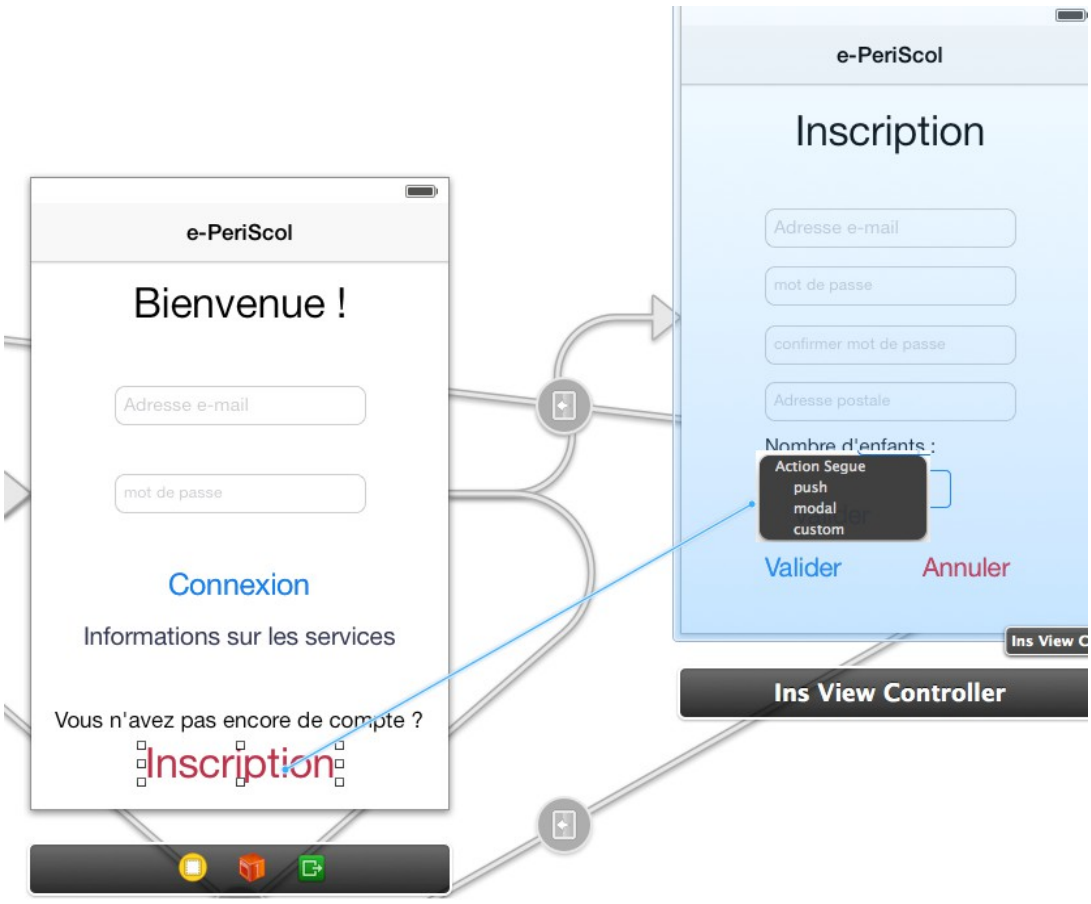
Pour arriver à ce résultat on créé une méthode, qui possède un composant de type **UIAlertView**. On édite le titre dans la partie **initWithTitle**, le contenu dans la section **message** et l'intitulé du bouton pour fermer dans **cancelButtonTitle**. Vu que l'on a pas besoin d'autres boutons on passe **nil** (= NULL) dans **otherButtonTitles**.

```
-(void) InfoService {
    UIAlertView *alert = [[UIAlertView alloc]initWithTitle: @"Services disponibles"
        message: @"Sport :\n Judo (5-18ans)\n Basketball (5-18ans)\n Football (5-18ans)
        \n\n Activités :\n Musique (5-18ans)\n Poterie (7-16ans)\n Dessin(12-16ans)"
        delegate:(self) cancelButtonTitle: @"OK" otherButtonTitles : nil];
    [alert show];
}
```

a.4. Bouton Inscription :

Le bouton **Inscription** envoi l'utilisateur à la page d'inscription. Pour cela on procède à un changement d'écran. On lie donc l'**IHM** de **Bienvenue** à l'**IHM** d'**Inscription**.

Les flèches entre les vues indiquent les liens entre elles. Ici nous nous intéressons au lien entre Bienvenue et Inscription. En faisant ce liens on peut décider de la transition. On choisis **push** qui fait glisser l'écran courant à gauche pour afficher le suivant.



b. Écran Inscription :

L'écran Inscription permet de créer un compte en donnant les informations nécessaire. L'adresse email, le mot de passe, l'adresse postale et le nombre d'enfant. Le bouton Valider créer le compte dans la BDD et ouvre la page **Accueil**, le bouton Retour renvoi à l'écran de **Bienvenue**. *(Pour les changements de vues cf. 4) a.4)*

c. Écran Accueil :

e-PeriScol
Retour
Accueil
(Affichera la photos et les activités correspondantes aux enfants inscrits sur ce compte et permettra les inscriptions)
Ajouter un enfant

L'Accueil affiche les enfant liés au compte et permet de les inscrire à une activité. Le bouton Ajouter un enfant mène à la page **Ajout d'enfant**. Le bouton Retour renvoi à la page de **Bienvenue**. *(Pour les changements de vues cf. 4) a.4)*

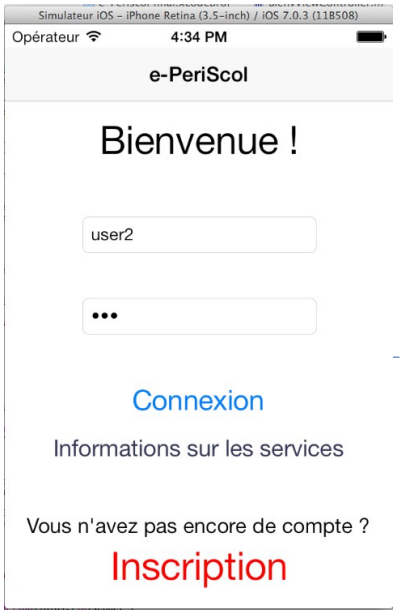
d. Écran Ajout Enfant :

L'écran Ajout d'un enfant permet de rajouter un enfant à un compte en rentrant son nom, son prénom, son sexe et sa date de naissance. Le bouton Valider entre les données de l'enfant dans la BDD. Le bouton Annuler renvoi à la page d'Accueil. *(Pour les changements de vues cf. 4). a.4)*

5) Test de validation :

1) Authentification :

Pour commencer le serveur doit bien être lancé :



```
15:47:39,178 Infos [org.apache.cxf.endpoint.ServerImpl] (MSC service thread 1-1) Setting the server's publish address to be htt
15:47:39,412 INFO  [org.jboss.wsf.stack.cxf.deployment.WSDLFilePublisher] (MSC service thread 1-1) WSDL published to: file:/C:/
15:47:39,422 INFO  [org.jboss.as.webservices] (MSC service thread 1-2) JBAS015539: Starting service jboss.ws.port-component-lin
15:47:39,425 INFO  [org.jboss.as.webservices] (MSC service thread 1-2) JBAS015539: Starting service jboss.ws.endpoint."E-perisc
15:47:39,430 INFO  [org.jboss.ws.common.management.DefaultEndpointRegistry] (MSC service thread 1-2) register: jboss.ws:context
15:47:39,453 INFO  [org.jboss.web] (MSC service thread 1-2) JBAS018210: Registering web context: /E-periscol
15:47:39,455 INFO  [org.jboss.as] (MSC service thread 1-1) JBAS015951: Admin console listening on http://127.0.0.1:9990
15:47:39,457 INFO  [org.jboss.as] (MSC service thread 1-1) JBAS015874: JBoss AS 7.1.1.Final "Brontes" started in 7687ms - Start
15:47:39,572 INFO  [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS018559: Deployed "Jboss_ServerEAR.ear"
15:47:39,573 INFO  [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS018559: Deployed "Jboss_Server.war"
15:47:39,573 INFO  [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS018559: Deployed "E-periscolEAR.ear"
15:47:39,574 INFO  [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS018559: Deployed "E-periscol.war"
15:47:39,864 INFO  [org.jboss.as.server.deployment.scanner] (MSC service thread 1-4) JBAS015012: Started FileSystemDeploymentSe
```

Ici l'utilisateur a entré «user2» comme email et «mdp» en mot de passe. Quand il appui sur le bouton Connexion on peut observer l'enveloppe **SOAP** que le serveur renvoi dans la console, on obtient.

```
<soap:Envelope xmlns:soap="http:/
   <soap:Body>
      <ns2:SeConnecterResponse xm
         <return>
            <Erreur>0</Erreur>
            <ID_Fam>103</ID_Fam>
            <Role>true</Role>
         </return>
      </ns2:SeConnecterResponse>
   </soap:Body>
</soap:Envelope>
```

On retrouve donc le code d'erreur 0 qui signifie que la connexion a bien été effectuée, ainsi que l'identifiant du compte et le rôle utilisateur du compte.

6) Bilan personnel :

Ce projet m'a permis de découvrir le travail sur **macOS**, l'environnement de développement **Xcode** ainsi que le langage **Objective-C**. Il m'a prouvé que je pouvais transférer ce que j'avais appris dans une technologie objet qui n'était pas étudiés en cours. Cela ma appris à suivre un cahier des charges mais aussi à le préciser en s'appuyant sur les différents cas d'utilisations envisageables. Pour finir ce projet a été l'occasion d'apprendre à travailler avec d'autres personnes et la bonne entente au sein de notre équipe a favorisée l'entraide.

X. Conclusion générale:

Toute l'équipe est satisfaite de ce projet, nous l'avons trouvé attrayant, motivant et éducatif.

Chacun d'entre nous à appris à travailler en équipe, en s'unissant et s'entraidant pour réaliser ce projet. Tout en y insufflant une ambiance conviviale propice au travail. De pars le plaisir de travailler ensemble au développement d'e-PeriScol. Ainsi que pour la satisfaction du travail collectif accompli.

De plus, l'équipe a acquis beaucoup de connaissances, grâce aux diversités des technologies présentes dans notre projet. Le renfort de nos compétences techniques, a contribué à notre épanouissement dans ce domaine.

Cette expérience bénéfique a convaincu la majorité d'entre nous de poursuivre leurs études, dans le but d'enrichir davantage leurs capacités à travers de nouveaux projets.